CHAPTER 7 – Using Object–Oriented JavaScript

Contents

[Introduction to OOP 1](#_Toc13663033)

[Understanding Encapsulation 2](#_Toc13663034)

[Encapsulated objects 2](#_Toc13663035)

[Interface 2](#_Toc13663036)

[Principle of information hiding 2](#_Toc13663037)

[Understanding Classes 2](#_Toc13663038)

[Using Build-In JavaScript Classes 3](#_Toc13663039)

[Instantiating an Object 4](#_Toc13663040)

[Performing garbage collection 4](#_Toc13663041)

[Using the Date, Number, and Math Classes 4](#_Toc13663042)

[Date Class 5](#_Toc13663043)

[Manipulating the Date and Time with the Date Class 5](#_Toc13663044)

[Number Class 8](#_Toc13663045)

[Math Class 9](#_Toc13663046)

[Defining Custom JavaScript Objects 10](#_Toc13663047)

[Declaring Basic Custom Objects 10](#_Toc13663048)

[Declaring Sub-Objects 11](#_Toc13663049)

[Adding Properties to an Existing Object 13](#_Toc13663050)

[Referring to Object Properties as Associative Arrays 13](#_Toc13663051)

[Creating Methods 15](#_Toc13663052)

[Enumerating Custom Object Properties 16](#_Toc13663053)

[Deleting Properties 17](#_Toc13663054)

[Defining Constructor Functions 17](#_Toc13663055)

[Adding Properties to a Constructor Function 18](#_Toc13663056)

[Adding Methods to a Constructor Function 18](#_Toc13663057)

[Using the prototype Property 20](#_Toc13663058)

## Introduction to OOP

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

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

### Understanding Encapsulation

#### Encapsulated objects

Objects are encapsulated, which means that all code and data are contained within the object itself. Encapsulation places code inside a “black box”.

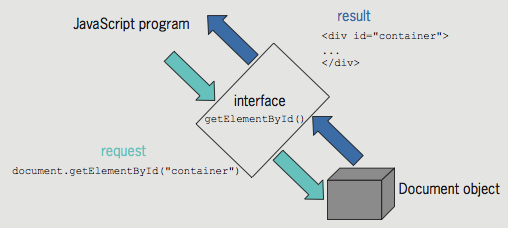
#### Interface

* + 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

Example:

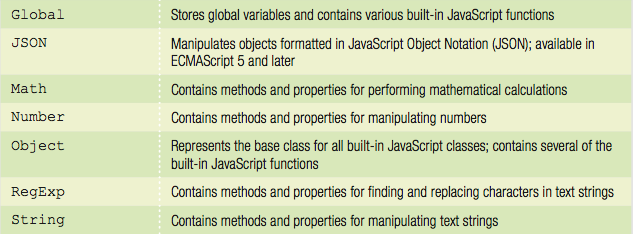
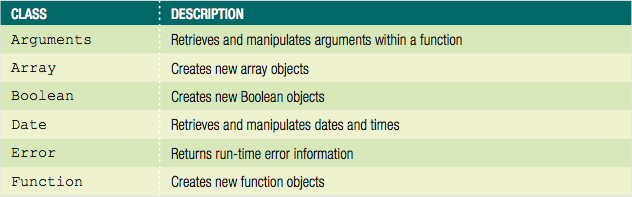
* Document object is encapsulated (black box)
  + getElementById() method
    - Part of the interface JavaScript uses to communicate with the Document object
* Microsoft Word: example of an object and its interface

### 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. For example, the Document object does not need to be instantiated to be used.

#### Using Build-In JavaScript Classes

For example, when you use the Array class you can use all of the methods and properties that come with it.



Example: You can use the Math class of the built-in JavaScript objects directly in your code, without instantiating a new object.

// 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;

}

#### Instantiating an Object

* + Can instantiate Array object using array literal

var deptHeads = [];

* + Can instantiate empty generic object using object literal

var accountsPayable = {};

* + - Generic object literal uses curly braces around value
  + Can't use object literal for Date object
    - Must use constructor

var today = new Date();

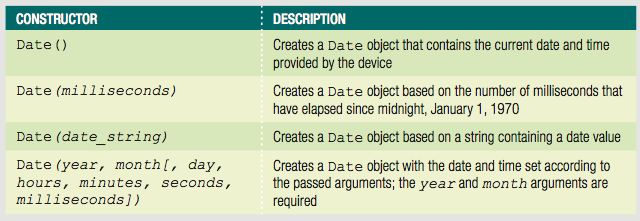
#### Performing garbage collection

* + 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

### Date Class

* + Methods and properties for manipulating the date and time
  + Allows use of a specific date or time element in JavaScript programs



Example:

var today = new Date();

* Days of the week and months of the year
  + Stored using numeric representations

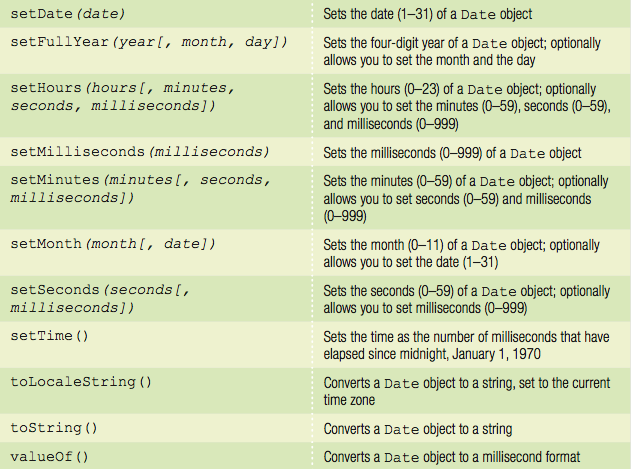
Starting with zero: similar to an array

Example:

var independenceDay = new Date(1776, 6, 4);

#### Manipulating the Date and Time with the Date Class

* 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



* 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

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";

}

* 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",↵

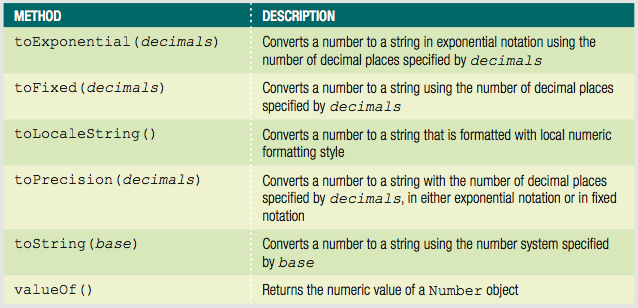
"July","August","September",↵

"October","November","December"];

var curMonth = months[today.getMonth()];

### Number Class

* Using Number class methods



Example:

var salesTotal = 1232345;

salesTotal = salesTotal.toLocaleString(); displays -> "1,232,345"

var salesTotal = 49.95;

var discount = salesTotal \* 0.1;

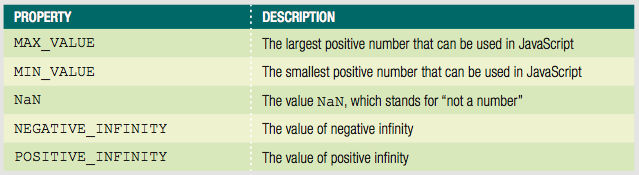
salesTotal -= discount;

44.955

salesTotal.toPrecision(2); displays -> "45"

salesTotal.toFixed(2); displays -> "44.95"

* Accessing Number class properties



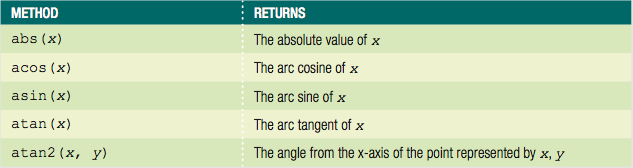
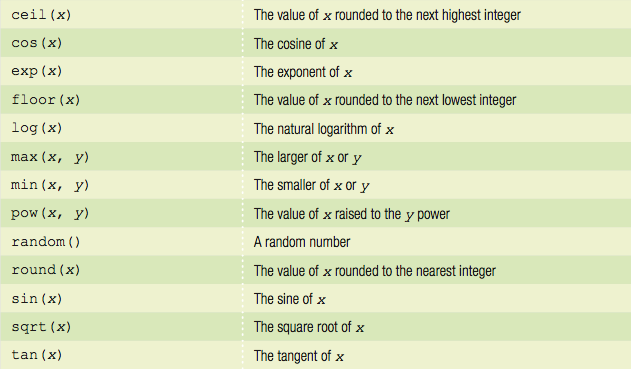
### Math Class

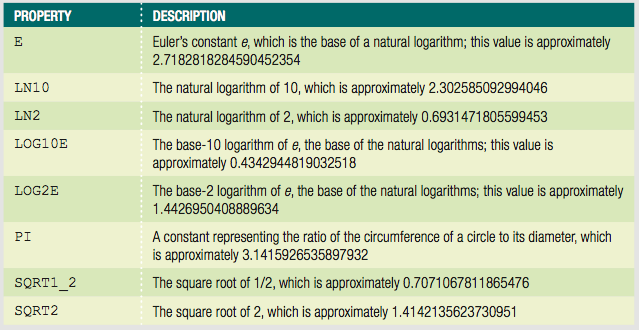
* Cannot instantiate a Math object using a statement such as: var mathCalc = new Math();
  + Use the Math object and one of its methods or properties directly in the code

Example:

var curNumber = 144;

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

Example:

var radius = 25;

var area = Math.PI \* Math.pow(radius,2);

var roundedArea = Math.round(area);

roundedArea displays -> 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

* You can create basic objects and properties by using the Object object.

You can declare a custom object with the Object() constructor –

var objectName = new Object();

It is easier to declare an object using an object literal –

var objectName = {};

* You can assign properties to the object using dot syntax.

var InventoryList = {};

InventoryList.inventoryDate = new Date(2019,01,31);

* You can declare an object and its properties all in a single statement with a literal.

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",

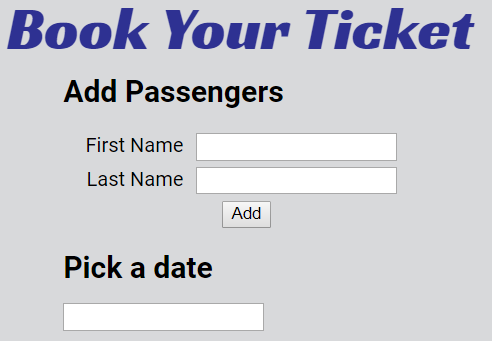
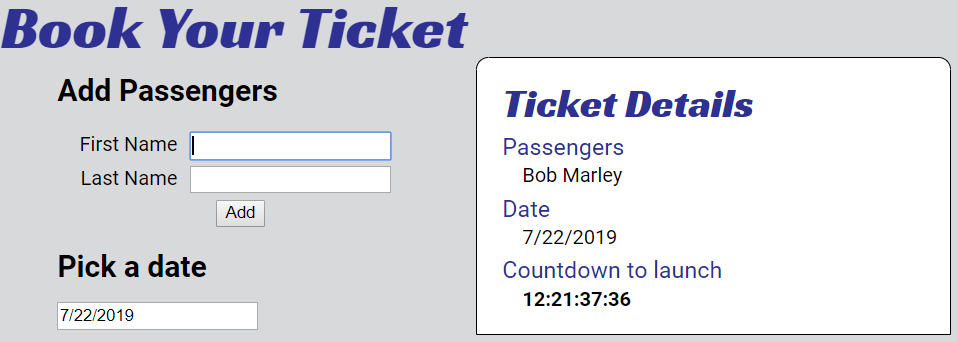
state: "NY",

zip: "14425"

}

};

EXAMPLE – Create a custom ticket object that contains passenger data:

<section id="ticket">

<h3>Ticket Details</h3>

<div id="passengersSection">

<h4>Passengers</h4>

<ul id="passengers"></ul>

</div>

<div id="dateSection">

<h4>Date</h4>

<p id="selectedDate"></p>

</div>

<div id="countdownSection">

<h4>Countdown to launch</h4>

<p id="countdown"></p>

</div>

</section>

<form>

<fieldset class="text">

<legend>Add Passengers</legend>

<label for="fname">First Name</label>

<input type="text" id="fname" />

<label for="lname">Last Name</label>

<input type="text" id="lname" />

<input type="button" value="Add" id="addName" />

</fieldset>

<fieldset>

<legend>Pick a date</legend>

<input type="date" id="tripDate" />

</fieldset>

<fieldset class="checks">

<legend>Choose a payment method</legend>

<input type="radio" id="single" name="Payment" />

<label for="single" id="singleLabel">Single payment</label>

<input type="radio" id="multiple" name="Payment" />

<label for="multiple" id="multipleLabel">60 monthly payments</label>

</fieldset>

</form>

#### Adding Properties to an Existing Object

Although you can declare properties within an object definition, it’s not required. As an alternative, you can add a new property simply by declaring its value. This is similar to the process of creating a new variable; however, you don’t use var or any other keyword to create a new object property.

For instance, to add a shippingSpeed property to the order object, you could use the following statement:

order.shippingSpeed = “overnight”;

#### Referring to Object Properties as Associative Arrays

JavaScript allows you to refer to object properties using associative array syntax.

* Associative array
  + An array whose elements are referred to with an alphanumeric key instead of an index number.
* With associative arrays
  + Can dynamically build property names at runtime

Example – if you stored the stoplight colors in an object, it might look as follows:

var stopLightColors = {

stop: "red",

caution: "yellow",

go: "green"

};

To refer to object property values using dot syntax:

stopLightColors.caution;

To refer to object property values using associative array:

stopLightColors["caution"];

To simply reference a property value, it’s almost always easier to use dot syntax. However, associative array syntax can be powerful when an object contains multiple properties with similar names that also incorporate numbers. For instance, an object containing items in a customer’s order might be structured as follows:

var order = {

item1: "KJ2435J",

price1: 23.95,

item2: "AW23454",

price2: 44.99,

item3: "2346J3B",

price3: 9.95

};

When creating an order summary, instead of referencing each property value with a separate statement, you could use a for loop like the following:

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>";

}

Associative array syntax also enables you to write generic code to add new object properties that incorporate numbers. For instance, if the customer with the order shown above decided to add an additional item to her shopping cart, your code could easily extend the order object with additional properties using the following code:

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);

#### Creating Methods

There are two ways to add a method to an object: by providing code for the method in the object, or by referencing an external function.

You can specify a method name with an anonymous function as its value. By assigning a method name to an anonymous function, you can then call the function using the method name. For instance, the following code creates an order object with a method named generateInvoice():

var order = {

items: {},

generateInvoice: function() {

// *function statements*

}

};

Even though the generateInvoice() method is created using an anonymous function, you can call the generateInvoice() method using the statement.

order.genrateInvoice;

Note : reference to existing function cannot have parentheses

Another way to declare a method is to specify a method name and use the name of an existing function as its value, as in the following code:

function processOrder() {

// *function statements*

}

var order = {

items: {},

generateInvoice: processOrder

};

#### 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);*

}

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

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();

}

}

The if statement refers to the prop variable as an index of the order object.

#### Deleting Properties

* Use the delete operator
* Syntax

delete *object.property*

* Example:

delete order.itemInStock;

#### Defining Constructor Functions

Another way to define your own custom objects is by using a constructor function, which is a function that is used as the basis for a custom object. JavaScript objects inherit all the variables and statements of the constructor function on which they are based.

Any JavaScript function can serve as a constructor.

***Best Practices –***

The main difference between creating an object with an object literal and using a constructor function is that the constructor function serves as a template, enabling you to create any number of objects with the same set of properties and methods defined in the constructor function.

The following code defines a function named Order() with four parameters that can serve as a constructor function:

function Order(number, order, payment, ship){

// statement(s);

}

You then use a var statement to instantiate an instance of the function, just as you do when using the constructor function for a build-in JavaScript object such as Date. For instance to create an instance of the Order object named shoppingBasket, you would use the following statement:

var shoppingBasket = new Order();

* If you need to create a unique object in a program, an object literal is the easiest solution.
* If your program will require multiple instances of an object with the same properties and methods, then creating a constructor function is more efficient.

#### Adding Properties to a Constructor Function

To add a property to a constructor function, you must add a statement to the function body that uses the **this** keyword with the following syntax:

this.property\_name = value;

In the case of a custom JavaScript object, the this keyword refers to the object that calls the constructor function.

For example, the following constructor function includes four properties: customerNumber, orderDate, paymentMethod, and shippingDate.

function Order(number, order, payment, ship) {

this.customerNumber = number;

this.orderDate = order;

this.paymentMethod = payment;

this.shippingDate = ship;

}

#### Adding Methods to a Constructor Function

You can create a function that will be used as an object method by referring to any object properties it contains with the this reference. For example, the following code defines a method that prints the customerNumber, orderDate, paymentMethod, and shippingDate properties of the Order constructor:

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>");

}

After a method is created, it must be added to the constructor function, using the syntax:

*this.methodname = functionname;*

The methodname following the *this* reference is the name that is being assigned to the function within the object.

The following code shows the Order() constructor function, the displayOrderInfo() function that creates the showOrder() method, and a statement that instantiates an Order object and prints the values of its properties.

function Order(number, order, payment, ship) {

this.customerNumber = number;

this.orderDate = order;

this.paymentMethod = payment;

this.shippingDate = ship;

this.showOrder = displayOrderInfo;

}

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>");

}

var shoppingBasket = new Order("KJ235I", new Date(2017, 6, 17),

"visa", new Date(2017, 6, 18));

shoppingBasket.showOrder();

#### Using the prototype Property

As explained earlier, an object inherits the properties and methods of the constructor function from which it is instantiated. After instantiating a new object, you can assign additional properties to the object, using dot syntax.

// constructor function

function Order(number, order, payment, ship){

this.customerNumber = number;

this.orderDate = order;

this.paymentMethod = payment;

this.shippingDate = ship;

}

// create a new object based on the Order constructor

var shoppingBasket = new Order("KJ230P", new Date(2017, 6, 17), "visa",

new Date(2017, 6, 18) );

// assigns to the new object a new property

shoppingBasket.trackingNumber = "Z20535550349";

When you add a new property to an object that has been instantiated from a constructor function, the new property is available only to that specific object; the property is not available to the constructor function or to any other objects that were instantiated from the same constructor function.

However, if you use the prototype property with the name of the constructor function, any new properties you create will also be available to the constructor function and any objects instantiated from it. The ***prototype property*** is a built-in property that specifies the constructor from which an object was instantiated.

By using the prototype property, you ensure that all objects that extend the Order constructor function also have access to the trackingNumber property.

// assigns to the new object a new property with prototype

shoppingBasket.prototype.trackingNumber = "Z20535550349";

Object definitions can use the prototype property to extend other object definitions. That is to say, you can create a new object based on an existing object. The new object inherits the properties and methods of the original object. You can then add additional properties and methods to the new object that will not be available to the existing object.

To extend one object definition from another object definition, you append the prototype property to the derived object definition, followed by the new keyword and the name of the base object definition using the following syntax:

*derived\_object.prototype = new base\_object();*

*Example* – A GraduationEvent object definition that extends the Event object definition. The Event class definition contains some generic properties that apply to all types of events, along with a method that calculates the cost of an event. The GraduationEvent class includes two properties with a showEventDetails method.

// Object definition - Event

function Event(location, date){

// properties

this.eventLocation = location;

this.eventDate = date;

// method

this.calcEventCost = calcCost;

this.eventCost = 0;

}

// method for Event object

function calcCost(guests){

this.eventCost = guests \* 25; //$25 per head

}

// Object definition - GraduationEvent

function GraduationEvent( graduate, school){

// properties

this.guestOfHonor = graduate;

this.schoolName = school;

// method

this.showEventDetails = eventDetails;

}

// method for GraduationEvent object

function eventDetails(){

var summaryDiv = document.getElementById("summarySection");

summaryDiv.innerHTML += ("<p>Guest of Honor: " + this.guestOfHonor + "</p>");

summaryDiv.innerHTML += ("<p>School: " + this.schoolName + "</p>");

summaryDiv.innerHTML += ("<p>Event date: " + this.eventDate + "</p>");

summaryDiv.innerHTML += ("<p>Event location: " + this.eventLocation +

"</p>");

summaryDiv.innerHTML += ("<p>Event cost: " + this.eventCost.toLocalString() +

"</p>");

}

// GraduationEvent object definition extends the Event object definition

GraduationEvent.prototype = new Event();

// instantiate a new object based on GraduationEvent constructor function

var smithGraduation = new GraduationEvent("Jacob Smith","Augusta Tech");

// use inherited properties and method

smithGraduation.eventLocation = "Santa Barbara, CA";

smithGraduation.eventDate = "May 27, 2017";

smithGraduation.calcEventCost(175);

// call method from the Graduation constructor funcion

smithGraduation.showEventDetails();