var Interface = function (objectName, methods) {

// Check that the right amount of arguments are provided

if (arguments.length != 2) {

throw new Error ("Interface constructor called with " + arguments.length + "arguments, but expected exactly 2.");

}

// Create the public properties

this.name = objectName;

this.methods = [];

// Loop through provided arguments and add them to the 'methods' array

for (var i = 0, len = methods.length; i < len; i++) {

// Check the method name provided is written as a String

if (typeof methods[i] !== 'string') {

throw new Error ("Interface constructor expects method names to be " + "passed in as a string.");

}

// If all is as required then add the provided method name to the method array

this.methods.push(methods[i]);

}

};

/\*

\* Adds a static method to the 'Interface' constructor

\* @param object | Object Literal | an object literal containing methods that should be implemented

\*/

Interface.ensureImplements = function (object) {

// Check that the right amount of arguments are provided

if (arguments.length < 2) {

throw new Error ("Interface.ensureImplements was called with " + arguments.length + "arguments, but expected at least 2.");

}

// Loop through provided arguments (notice the loop starts at the second argument)

// We start with the second argument on purpose so we miss the data object (whose methods we are checking exist)

for (var i = 1, len = arguments.length; i < len; i++) {

// Check the object provided as an argument is an instance of the 'Interface' class

var interface = arguments[i];

if (interface.constructor !== Interface) {

throw new Error ("Interface.ensureImplements expects the second argument to be an instance of the 'Interface' constructor.");

}

// Otherwise if the provided argument IS an instance of the Interface class then

// loop through provided arguments (object) and check they implement the required methods

for (var j = 0, methodsLen = interface.methods.length; j < methodsLen; j++) {

var method = interface.methods[j];

// Check method name exists and that it is a function (e.g. test[getTheDate])

// if false is returned from either check then throw an error

if (!object[method] || typeof object[method] !== 'function') {

throw new Error ("This Class does not implement the '" + interface.name + "' interface correctly. The method '" + method + "' was not found.");

}

}

}

};

/\*

Next, in the HTML file we have an inline script that creates a new Interface instance and

then passes it the methods we are expecting. The user of the API would need to implement

an object literal of all the methods the interface requires (note the following code could

be in a separate SCRIPT file as long as it is run after the above script):\*/

// We pass into the Interface the name of the current Interface instance,

// followed by an Array of the methods we are expecting to find

var test = new Interface('test', ['details', 'age']);

var properties = {

name: "Jay Roy",

actions: {

details: function() {

return "I am " + this.age() + " years old.";

},

age: (function(birthdate) {

var temp=birthdate.split(',')

var dob = new Date(temp[0],temp[1], temp[2]),

today = new Date(),

ms = today.valueOf() - dob.valueOf(),

minutes = ms / 1000 / 60,

hours = minutes / 60,

days = hours / 24,

years = days / 365,

age = Math.floor(years)

return function() {

return age;

};

})("1980, 05, 31")

}

};

// Create a Person constructor that will implement the above properties/methods

function Person(config) {

// Pass in the methods we are expecting,

// followed by the name of the Interface instance that we're checking against

Interface.ensureImplements(config.actions, test);

this.name = config.name;

this.methods = config.actions;

}

// Create a new instance of the Person constructor...

var me = new Person(properties);

// ...and make sure the methods are working

console.log(me.methods.age());

console.log(me.methods.details());