INTRODUCTION TO JAVASCRIPT OBJECTS

Topics

Object-oriented programming in JavaScript

- Factory Pattern
- Constructor Pattern
- Prototype Pattern
- Dynamic Prototype Pattern
Classless Objects

JavaScript has no class definitions.
Objects are created based on the language-defined `Object` construct.
To create multiple objects based on a desired object structure, functions written to guide instantiation.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-1.htm

JavaScript's class-free object-oriented structure is a frequently criticized element.

Factory Pattern

One way to create multiple objects with the same properties and functions is to use the Factory Pattern design pattern.

Function takes in property values, creates an object, assigns properties and functions to the object, and returns the object.
The function becomes an "object factory".

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-2.htm

Use identifiers beginning with capital letters for objects (only) to denote distinction from identifiers used with primitive data types.
Another alternative is the **Constructor Pattern**. With this pattern, an object is created (using new) in the function call.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-3.htm

The **constructor function name** is the name of the desired object.

**this** is used in parameter/property and function assignment.

---

**instanceof**

**instanceof** is a JavaScript operator used to see if a particular object is an instance of (an instantiation of) a particular object pattern.

An object created with the Constructor Pattern is an **instanceof** both Object and the specified constructor name.

An object created with the Factory Pattern is an **instanceof** Object only.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-3a.htm
Prototype Pattern

Another alternative is the **Prototype Pattern**. With this pattern, memory and resources are used more efficiently as multiple objects share resources.

The Constructor function body is (typically) null.

Properties and functions are assigned to the constructed object's `prototype`.

`prototype` is a default property within an object that contains properties and methods available to all instances of that object.

This technique allows defining 'default values' for objects and object functions. The object creation call does not specify property values.

[http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-4.htm](http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-4.htm)

---

Prototype pattern

With the Prototype Pattern, value assignments to the **instantiated** (or **instance**) object `shadow` property values in the prototype (they do not replace them). (Creating a 'search order'.)

`delete` can be used to remove a property/value pair from an object.

[http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-5.htm](http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-5.htm)
Comparing the patterns

**Factory Pattern and Constructor Pattern**
- Object
  - Properties
  - Functions

**Prototype Pattern**
- Prototype
  - Properties
  - Functions

---

toString

Each object contains a language-defined function **toString** which outputs the object as a String.

Unless overridden in our object instance, this will output "$[object Object]".

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-5a.htm

This function can/should be overridden based on our desire of how the object should be represented as a string.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-5b.htm
Test your skills

In the prototype or the instance?

The **in operator** checks to see whether a specified property is present within an object.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-6.htm

`object.hasOwnProperty("x")` checks whether property `x` is present in the object *instance*. (i.e. it is not there because of the prototype)

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-7.htm
Checking for, enumerating properties

The **for...in loop** can be used to enumerate all properties and functions present in an object.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-8.htm

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-9.htm

http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-10.htm

Ordering of properties within an object is not fixed.

Test your understanding
Property enumeration

When properties of an object are enumerated, some properties are not listed.
Some built in properties are designated as non-enumerable. Examples include toString and others.
Browsers are inconsistent in their behavior of these non-enumerable elements. In IE, if you override the non-enumerable property with your own, your override is still non-enumerable. In Opera, Safari, and Firefox the behavior varies depending on property.
http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-11.htm

Dynamic Prototype Pattern

2 notable disadvantages of using the "pure" Prototype Pattern:
1. Cannot instantiate and set properties in one line.
2. As all instances are references to the prototype, if the prototype contains an object, this single object is shared among all instances.
Solution: use a combination of the Constructor and Prototype Patterns—the **Dynamic Prototype Pattern**.
Properties set in the Constructor pattern fashion. Each function set on the prototype *if not already in existence.*
http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-12.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/6-13.htm
Object Patterns

Benefit of object patterns—can learn syntax/pattern and reuse with confidence.
Other object patterns exist of varying complexity.

Topics

Object-oriented programming in JavaScript
- Factory Pattern
- Constructor Pattern
- Prototype Pattern
- Constructor/Prototype Pattern