

OOP patterns

Ilya Kantor

General concepts

JavaScript is a very flexible language. In contrast with Java, PHP, C++ and many other languages, there are many ways to implement OOP in JavaScript.

[Read more »](#)

Pseudo-classical pattern

1. [Pseudo-class declaration](#)
2. [Inheritance](#)
3. [Calling superclass constructor](#)
4. [Overriding a method \(polymorphism\)](#)
 1. [Calling a parent method after overriding](#)
 2. [Sugar: removing direct reference to parent](#)
5. [Private/protected methods \(encapsulation\)](#)
6. [Static methods and properties](#)
7. [Summary](#)

In pseudo-classical pattern, the object is created by a constructor function and it's methods are put into the prototype.

Pseudo-classical pattern is used in frameworks, for example in Google Closure Library. Native JavaScript objects also follow this pattern.

[Read more »](#)

All-in-one constructor pattern

1. [Declaration](#)
2. [Inheritance](#)
3. [Overriding \(polymorphism\)](#)
4. [Private/protected methods \(encapsulation\)](#)
5. [Summary](#)
 1. [Comparison with pseudo-classical pattern](#)

All methods and properties of the object can be added in the constructor. This method doesn't use prototype at all.

[Read more »](#)

Factory constructor pattern

1. [Declaration](#)
2. [Inheritance](#)
3. [Private/protected methods \(encapsulation\)](#)

This pattern is special, because it doesn't use "new". The object is created by a simple function call, similar to *Python*-style:

```
var animal = Animal("fox")
var rabbit = Rabbit("rab")
```

4. [Summary](#)

1. [Comparison with All-in-one constructor](#)



Read more »