Jest wiele sposobów na tworzenie obiektów w JS . Każdy ma jakieś wady i zalety. Należy wybrać taki, który nam się najbardziej podoba, i który akurat przy konkretnym zastosowaniu jest najwygodniejszy. W tym artykule pokażę, jak tworzyć w JS obiekty z wykorzystaniem mechanizmu prototypowania.

Język obiektowy bez klas

No tak! Ledwo zacząłeś widzieć korzyści płynące z tworzenia klas i operowaniu na obiektach, a tu Ci ktoś mówi, że nie ma klas w JS:). Ano nie ma. Są za to prototypy. I tak naprawdę to zasada jest bardzo podobna. Zobaczmy kod, bo tak zawsze łatwiej:

{codecitation class='brush: js'}function MyObject() { } // 1 MyObject.prototype.sayHi = function() // 2 { alert("Witaj świecie"); // 3 }; var oMyObj = new MyObject(); // 4 oMyObj.sayHi() // 5{/codecitation}

1. Tworzymy pustą funkcję. Jako, że w JS wszystko jest obiektem, to funkcja też jest obiektem. Nazwiemy takie "cuś" konstruktorem.
2. Do prototypu obiektu o nazwie MyObject dodajemy metodę `sayHi()', ...
3. ... która po prostu wyświetla "witaj świecie"
4. Tworzymy nowy obiekt. Zauważ, że używamy słówka (fachowo - operatora) `new'. W tym momencie do zmiennej oMyObj zostaje przypisany obiekt stworzony wg wytycznych zawartych w prototypie MyObject. Czyli posiada pusty konstruktor (patrz [1]) oraz metodę `sayHi()' [2]. Konstruktor zostaje wywoływany przy tworzeniu obiektu. Metodę, możemy wyołać sami, co też robimy w [5].
5. Wywołujemy metodę sayHi().

Po wykonaniu tego skryptu w przeglądarce powinnien pojawić się alert z napisem "witaj świecie".

Właściwości prototypu obiektu

Obiekt może mieć właściwości i metody (w uproszczeniu: zmienne = właściwości, funkcje = metody). W językach posiadających klasy właściwości są przypisywane do klasy. W JS można to zrobić na wiele sposobów (porównaj artykuły o dodawaniu metod do istniejących obiektów oraz o rozszerzaniu prototypu obiektu). Stwórzmy zatem taką właściwość, którą będą posiadać wszystkie obiekty tworzone wg prototypu obiektu MyObject:

{codecitation class='brush: js'}function MyObject() {} MyObject.prototype.sentence = 'Witaj świecie'; // 1 MyObject.prototype.sayHi = function() { alert(this.sentence); // 2 }; var oMyObj = new MyObject(); oMyObj.sayHi();{/codecitation}

Nic się nie zmieniło w wyniku, porównując z poprzenim przypadkiem. Jednak w kodzie widzimy dwie zmiamy:

1. Do prototypu obiektu MyObject dodajemy właściwość `sentence' i przypisujemy jej wartość "Witaj świecie".
2. W alercie nie wstawiamy już na sztywno ciągu znaków, a odnosimy się do właściwości `sentence'. Używamy tu operator `this'. Można to przetłumaczyć tak: "jestem obiektem stworzonym wg prototypu `MyObject'. `this' to ja sam. Poszukaj zatem mojej właściwości o nazwie `sentece'". W językach, w których są klasy [np. PHP] także występuje operator $this.

Skoro już wyrzuciliśmy powitalny tekst do zmiennej, to spróbujmy go zmienić :). Najlepiej za pomocą konstruktora. Co on taki pusty ma być :)

Sparametryzowanie konstruktora

{codecitation class='brush: js'}function MyObject(msg) // 1 { if (msg) // 2 { this.sentence = msg; // 3 } } MyObject.prototype.sentence = 'Witaj świecie'; // 4 MyObject.prototype.sayHi = function() { alert(this.sentence); // 5 }; var oMyObj_one = new MyObject(); // 6 var oMyObj_two = new MyObject("A ja nie będę taki jak inni"); // 7 var oMyObj_three = new MyObject(); // 6 oMyObj_three.sentence = 'Za to ja to już w ogóle jest krejzol'; // 8 oMyObj_one.sayHi(); // 9 oMyObj_two.sayHi(); // 9 oMyObj_three.sayHi(); // 9{/codecitation}

Się nam kodu narobiło :). Spokojnie, nie jest tak źle jakby się mogło wydawać. Po kolei:

1. Nasz konstruktor posiada teraz parametr. Czyli tworząc nowy obiekt możemy podać mu jakąś wartość.
2. Ten warunek można przetłumaczyć na: "Jeśli `msg' ma jakąś wartość". Czyli jeśli wywołamy ten konstrukotr bez parametrów (patrz [6]) to nie wejdzie do tego ifa. Czytaj więcej o fasy values .
3. Jeśli podaliśmy jakiś ciąg jako parametr (patrz [7]) to do właściwości `sentence' tego konkretnego obiektu zostanie przypisana nowa wartość i ...
4. ... nadpisze ona domyślną wartość pola `sentece'.
5. W metodzie `sayHi()' wyświetlone zosatnie zdanie podane do konstruktora, lub domyślna wartość jeśli do konstruktora nie podano zdania.
6. Tworzymy obiekty bez parametrów. Czyli w polach `sentece' tych obiektów występuje wartość domyślna.
7. Tworzymy obiekt podając konkretną wartość do konstruktora. W tym obiekcie w polu `sentence' jest wartość "A ja nie będę taki jak inni".
8. Nadpisujemy "ręcznie" wartość właściwości `sentence'.
9. Wywołujemy metodę `sayHi()' wszystkich trzech obiektów.

Ja rzadko używam tego sposobu tworzenia obiektów w JS, mimo, że jest on najbardziej klasopodobny. Staram się raczej wykorzystywać wzorzec modułu .

Pełen kod z dodatkowym bonusem:

Jak sądzisz, co zrobi kod z framentu oznacznonego BONUS?

{codecitation class='brush: html'}<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">

<head>

<title>Tworzenie obiektów z wykorzystaniem prototypów</title>

<meta http-equiv="content-type" content="text/html;charset=utf-8" />

<script type="text/javascript">

(function() {

function MyObject() {}

MyObject.prototype.sayHi = function()

```
    {

        alert(&amp;quot;Witaj świecie&amp;quot;);

    };

    var oMyObj = new MyObject();

    oMyObj.sayHi();

})();



(function() {



    function MyObject() {}

    MyObject.prototype.sentence = 'Witaj świecie';
```

```
    MyObject.prototype.sayHi = function()

    {

        alert(this.sentence);

    };

    var oMyObj = new MyObject();

    oMyObj.sayHi();

})();



(function() {

    function MyObject( msg )

    {

        if (msg)
```

```
    {

        this.sentence = msg;

    }

}

MyObject.prototype.sentence = 'Witaj świecie';

MyObject.prototype.sayHi = function()

{

    alert(this.sentence);

};
```

```
    var oMyObj_one = new MyObject();

    var oMyObj_two = new MyObject(&amp;quot;A ja nie będę taki jak inni&amp;quot;);

    var oMyObj_three = new MyObject();

    oMyObj_three.sentence = 'Za to ja to już w ogóle jest krejzol';

    oMyObj_one.sayHi();

    oMyObj_two.sayHi();

    oMyObj_three.sayHi();

})();

 /* BONUS  */

(function() {

    function MyObject( msg )
```

```
{

    if (msg)

    {

        this.sentence = msg;

    }

}

MyObject.prototype.sentence = 'Witaj świecie';




MyObject.prototype.sayHi = function()

{

    alert(MyObject.prototype.sentence);
```

```
};

var oMyObj_one = new MyObject();

var oMyObj_two = new MyObject(&amp;quot;A ja nie będę taki jak inni&amp;quot;);

var oMyObj_three = new MyObject();

oMyObj_three.sentence = 'Za to ja to już w ogóle jest krejzol';

oMyObj_one.sayHi();

oMyObj_two.sayHi();

oMyObj_three.sayHi();

})();

</script>

</head><body></body></html>{/codecitation}
```

Jeśli nie rozumiesz o co chodzi z takim kodem:

{codecitation class='brush: js'}(function() { ... })();{/codecitation}

To warto przeczytać artykuł o zasięgu zmiennych w JS . Szczególnie część o "Rozsądnym zastosowaniu".