Wzorzec modułu różni się bardzo niewiele od wzorca fabryki , który opisywałem wcześniej. na dobrą sprawę można wręcz uznać, że to jest to samo.

Chciałbym zatem tu pokazać kilka dobrych praktyk jakich warto używać przy tworzeniu obiektów z wykorzystaniem wzorca fabryki / modułu. Cześć z nich jest nieosiąglana w przypadku stosowania np. tworzenia obiektów z prototypu.

1. Pola prywatne

Można tworzyć obiekt tak:

```
{codecitation}function creator() {
        var obj = {};
        obj.func1 = function() {
                // tu cos robimy;
        }
        obj.func2 = function() {
                return obj.func1() + 2;
        }
        return obj;
 };{/codecitation}
```

Jednak wtedy możemy zrobić coś takiego:

```
 {codecitation}var o = creator(); // w zmiennej o mamy obiekt z metodami func1 i func2
 o.func1 = function() {
        // kod robiuacy cos zupelnie innego niz pierwotna funkcja
 }
 o.func2(); // wynik zupelnie inny niż pierwotnie!{/codecitation}
```

Tak, łatwo krytykować, ale co zrobić aby tak nie było?

{codecitation}function creator() {
	function f1() {
		// tu cos robimy;
	}
	function f2() {
		return f1() + 2;
	}
	return {
		func1 : f1,
		func2 : f2
	};
};{/codecitation}

Dzięki temu nadpisując metodę func1, func2 nadal działa tak jak zakładaliśmy.

2. Używanie getterów / setterów

Czym jest getter / setter? Getter to metoda **pobierająca wartość** jakiejś zmiennej, a setter to metoda **ustawiająca wartość** zmiennej. Czemu nie odczytywać tego w ten sposób:

{codecitation}obj.a = 12;
alert(obj.a);{/codecitation}

Przyczyną jest to, że uzależniamy się od struktury naszego obiektu. Załóżmy, że mamy obiekt do obsługi kwadratu:

{codecitation}function kwadrat() {
	return {
		pole : 0,
		bok : 0,
		obwod : 0
	};

```
}
var kw = kwadrat();
kw.bok = 10; // ustawilismy dlugosc boku
alert(kw.pole); // nadal 0!!!{/codecitation}
```

A jakbyśmy zrobili to poprawnie:

```
{codecitation}function kwadrat() {
    var   pole_powierzchni = 0,
          dlugosc_boku = 0,
          dlugosc_obwodu = 0;
    function pole( p ) {
        if (!p) {
            return pole_powierzchni;
        }
        pole_powierzchni = p;
        dlugosc_boku = Math.sqrt(p);
        dlugosc_obwodu = 4*dlugosc_boku;
    }
    function bok( b ) {
        if (!b) {
            return dlugosc_boku;
        }
        dlugosc_boku = b;
        pole_powierzchni = b*b;
        dlugosc_obwodu = 4*b;
    }
    function obwod( o ) {
        if (!o) {
            return dlugosc_obwodu;
        }
        dlugosc_obwodu = o;
        dlugosc_boku = o/4;
        pole_powierzchni = dlugosc_boku*dlugosc_boku;
    }
    return {
        pole : pole, // tu nazwy nie muszą się pokrywać
        bok : bok,
        obwod : obwod
    };
}
var kw = kwadrat();
kw.bok(10); // ustawilismy dlugosc boku
alert(kw.pole()); // 100{/codecitation}
```

Później może się okazać, że chcesz jeszcze coś pozmieniać. Gmerasz w "bebechach" ile chcesz. Ale interfejs pozostaje ciągle taki sam dla programisty - użytkownika. Prosty i przejrzysty. Taka zasada odnosi się nie tylko do JS. W każdym języku staraj się tak programować .

3. Konwencja

Coraz częściej spotykana zasada w programowaniu. Zamiast ustawiać wiele zmiennych, zamiast ciągle sprawdzać jak się coś zrobiło w którymś miejscu przyjmij odpowiednią konwencję. W JS nie widać która zmienna skąd pochodzi (zmienne globlane i lokalne wyglądają tak samo). Ale przecież nazwa może ci mówić, co to za zmienna. Oto kilka zasad, których używam ja (nie muszą być najlepszym rozwiązaniem, jednak nawet niedoskanała konwencja jest lepsza od braku jakiejkolwiek):

- pola prywatne obiektu zapisywane w notacji węgierskiej (możesz oczywiście w inny sposób je oznaczać). Dzięki temu od razu widzę, że to są tego typu zmienne i znam ich zasięg.

- jeśli używam słówka kluczowego function do tworzenia obiektu, to zawsze stosuję zapisu: {codecitation}
// przypisuje referencje (adres w pamięci) funkcji która tu tworzy obiekt do zmiennej (var obj)
var obj = function() {}; // srednik na koncu - wtedy wiem, ze ta klamra zamyka obiekt, a nie np. for{/codecitation} jeśli obiekt wykorzystuje operator new to jego nazwa powinna zaczynać się wielką literą. Jeśli nie - małą. We wzorcu modułu nie trzeba wykorzystywać operatora new (choć wywołanie funkcji tworzącej z operatorem new nie powoduje błędu).

- zmienne lokalne w metodach nazywam wykorzystujac notację "podkreśleniową" (nie wiem, jak się to nazywa po polsku:P), np.: {codecitation}function przyklad1() {
var zmienna_lokalna;
zmienna_lokalna = sZmiennaPrywatnaObiektu; // od razu widać skąd pochodzą dane!
}{/codecitation}
- koduj po polsku albo po angielsku. Staraj się unikać hybryd w stylu: {codecitation}var zmienna = my_function(literka, char, ilewynikow, count){/codecitation} taki kod wyglada na bardzo nieprofesjonalny. Dodatkowo ciezko sie go czyta. Nie ma nic zlego w nazywaniu metod i zmiennych polskimi nazwami. Jednak z głowa!
- ostatnie, ale nie najmniej ważne: niechże kod sam sie komentuje. Ja tu w przykładach

często używam zmiennej a, b, c itp. Jednak to dlatego, że ten kod nie ma większego sensu prócz czysto teoretycznego zastosowania pewnej konstrukcji. Kiedy już kodujesz, unikaj zmiennej o nazwie a (chyba, że to współczynnik przyspieszczenia, albo zmiennea przechowujaca węzeł DOM a).