Kurs frontend

Hello!

Tomasz Koszykowski

Fullstack javascript developer

Programowanie obiektowe

1. Idea programowania obiektowego

Idea programowania obiektowego
Definicja

Paradygmat programowania, w którym programy definiuje się za pomocą obiektów – elementów łączących stan (właściwości) i zachowanie (metody). Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

Idea programowania obiektowego
Abstrakcja

Polega na ukrywaniu lub pomijaniu mało istotnych informacji a skupieniu się na wydobyciu informacji, które są niezmienne i wspólne dla pewnej grupy obiektów.

Idea programowania obiektowego
Dziedziczenie

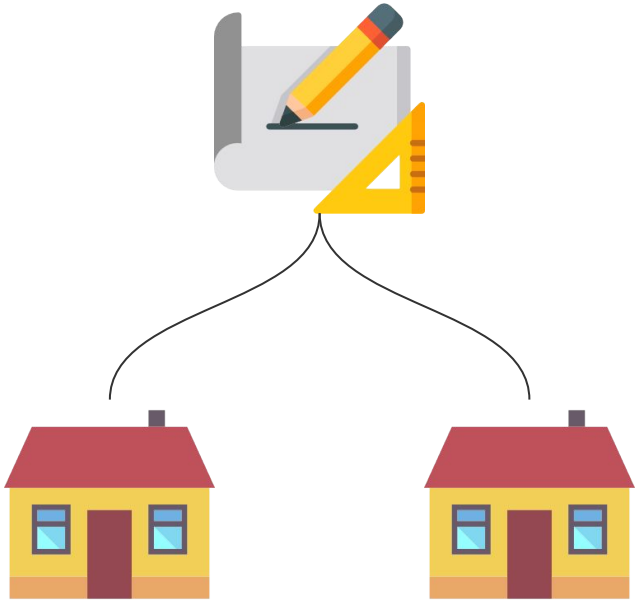
Definiowanie i tworzenie specjalizowanych obiektów na podstawie bardziej ogólnych.

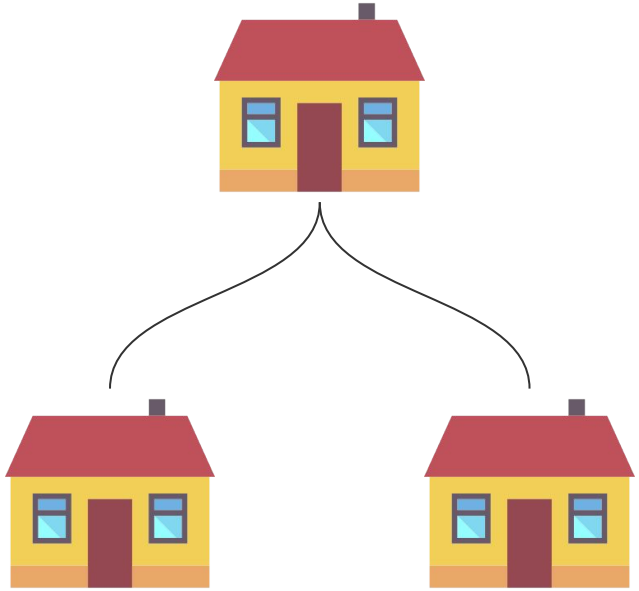
Dziedziczenie prototypowe - obiekty dziedziczą z innych obiektów.
Dziedziczenie klasowe - obiekty tworzone w oparciu o klasy.

W js mamy do czynienia z dziedziczeniem prototypowym.

Idea programowania obiektowego

Dziedziczenie klasowe vs prototypowe





Icons are from https://www.flaticon.com/

Idea programowania obiektowego
Enkapsulacja/Hermetyzacja

Ukrywanie implementacji. Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób.

Tylko własne metody obiektu są uprawnione do zmiany jego stanu.

Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy.

Idea programowania obiektowego
Polimorfizm

Możliwość wyabstrahowania wyrażeń od konkretnych typów.

Podczas pisania programu wygodnie jest traktować różne dane w jednolity sposób. Niezależnie czy należy wydrukować liczbę czy napis, czytelniej jest gdy operacja taka nazywa się po prostu drukuj, a nie drukuj_liczbę i drukuj_napis. Jednak napis musi być drukowany inaczej niż liczba, dlatego będą istniały dwie implementacje polecenia drukuj, ale nazwanie ich wspólną nazwą tworzy wygodny abstrakcyjny interfejs niezależny od typu drukowanej wartości.

Idea programowania obiektowego

Plusy i minusy

+ Przejrzystość
+ Łatwa modyfikacja kodu
+ Łatwe utrzymanie, rozbudowa oraz optymalizacja
+ Małe ryzyko utraty funkcjonalności i błędów

- Spora ilość nadmiarowego kodu
- Szybkość działania w porównaniu do kodu strukturalnego
- Wymagana dokładna i szczegółowa analiza przy tworzeniu klas/prototypów

Idea programowania funkcyjnego
Porównanie z oop

Programowaniu funkcyjne charakteryzuje się dużą ilościa małych prostych funkcji.

Do osiągnięcia korzyści podobnych do dziedziczenia używa się kompozycji.

Głównym plusem jest prostota kodu.

W js efektywnie można łączyć programowanie obiektowe z funkcyjnym.

2. Kontekst wywołania funkcji

Kontekst wywołania funkcji
Słowo kluczowe this

Najczęściej this wskazuje na referencję do obiektu, który **wywołał** daną funkcję.

Globalnym kontekstem (this) w przeglądarce jest obiekt window.

Do zmiany kontekstu (nadpisania this) służą funkcje Function.prototype.bind /call /apply

Funkcje strzałkowe zachowują this z momentu **deklaracji,** a nie wywołania jak zwykłe funkcje. Funkcjom strzałkowym nie można nadpisać this.

Zadanie 0

Sprawdź na co wskazuje this w zasięgu globalnym oraz w funkcji.

Przypisz tę funkcję do obiektu i sprawdź this.

Kontekst wywołania funkcji
Function.prototype.call

Umożliwia wywołanie funkcji z podanym kontekstem (this)

```
function returnX () {
    return this.x;
}

var obj = { x: 42 }

returnX.call(this, 1, 2, 3) // undefined
returnX.call(obj, 1, 2, 3) // 42

returnX.call({}, 1, 2, 3) // undefined
returnX.call({ x: 142 }, 1, 2, 3) // 142
```

Zadanie 1

Użyj call do zmiany kontekstu wywoływanej funkcji

Przykaż do funkcji parametry

Kontekst wywołania funkcji
Function.prototype.apply

Umożliwia wywołanie funkcji z podanym kontekstem (this). Jak call, ale parametry podajemy jako tablica.

```
function returnX () {
    return this.x;
}

var obj = { x: 42 }

returnX.call(this, [1, 2, 3]) // undefined
returnX.call(obj, [1, 2, 3]) // 42

returnX.call({}, [1, 2, 3]) // undefined
returnX.call({ x: 142 }, [1, 2, 3]) // 142
```

Zadanie 2

Użyj apply do zmiany kontekstu wywoływanej funkcji

Przykaż do funkcji parametry

Kontekst wywołania funkcji
Function.prototype.bind

Zwraca nową funkcję z przypisanym kontekstem (this) przekazanym jako argument.

```
function returnX () {
    return this.x;
}

var obj = { x: 42 }

returnX.call(this, [1, 2, 3]) // undefined
returnX.call(obj, [1, 2, 3]) // 42

returnX.call({}, [1, 2, 3]) // undefined
returnX.call({ x: 142 }, [1, 2, 3]) // 142
```

Zadanie 3

Użyj bind do zmiany kontekstu wywoływanej funkcji

Przykaż do funkcji parametry

3. Tworzenie obiektów i dziedziczenie

Tworzenie obiektów i dziedziczenie

Object literal

Najprostszy sposób na stworzenie obiektu

```
var cat = {
    name : “Fluffy”,
    age: 1,
    sound: “Meeeeow!”,
    makeSound: function(){ console.log(this.sound) },
    speak: function(){
        console.log(‘Sorry cats can’t speak’)
    }
}
```

Object creation and inheritance in JS

Factory functions

Factory function na podstawie argumentów zwraca nowy obiekt.

```
function catFactory(name, age) {
    return {
        name : name,
        age: age,
        sound: “Meeeeow!”,
        makeSound: function(){ console.log(this.sound) },
        speak: function(){
            console.log(‘Sorry cats can’t speak’)
        }
    }
}
```

Zadanie 4

Stwórz factory function która tworzy obiekt cat z właściwością name

Stwórz kilka obiektów cat i przypisz je do zmiennych

Zadanie 5

Rozszerz funkcję z poprzedniego zadania.

Przekaż property sound, metoda make sound powinna wypisywać this.sound.

Tworzenie obiektów i dziedziczenie

What is prototype in JS?

Język JavaScript, w odróżnieniu od innych języków nie posiada pojęcia klasy. Tutaj wszystko jest obiektem, także funkcje, a zamiast dziedziczenia opartego na hierarchii klas mamy dziedziczenie prototypowe.

Na końcu łańcucha prototypów jest zawsze Object, którego prototypem jest null

Podczas gdy jest to często uważane za słabość języka JavaScript, prototypowe podejście do dziedziczenia jest w rzeczywistości znacznie potężniejszym narzędziem niż model klasowy. Dla przykładu trywialnie proste jest zbudowanie klas w modelu prototypowym, podczas gdy odwrotna operacja jest znacznie bardziej skomplikowana.

Tworzenie obiektów i dziedziczenie

__proto__ and prototype

__proto__ jest to wewnętrzna właściwość, której silnik js używa do trzymanie referencji do prototypu obiektu. Nie powinno się jest modyfikować manualnie.

prototype jest właściwością, którą posiadają wszystkie funkcje. Kiedy funkcja jest używana jako konstruktor (new Fn()) obiekt trzymany w prototype jest przypisywany jako __proto__ nowo powstałego obiektu.

```
const arr = new Array();
Array.prototype === arr.__proto__ //true
```

Tworzenie obiektów i dziedziczenie
Object.create()

Object.create() tworzy nowy obiekt, używając obiektu przekazanego w argumencie jako prototypu.

```
var base = { baseProperty: function(){ console.log('I'm from prototype!') } }

var obj = Object.create(base)

console.log(obj) // it's empty object? -> {}

obj.baseProperty() // I'm from prototype!
```

Zadanie 6

Stwórz obiekt cat, który ma metodą wypisującą this.sound. Stwórz nowy obiekt przy użyciu Object.create z obiektem cat jako prototyp i obiektem z właściwością sound. Wywołaj funkcję która wypisuje this.sound.

Tworzenie obiektów i dziedziczenie

Object.assign()

Object.assign() kopiuje wszystkie wyliczalne własne właściwości z jednego lub więcej obiektów źródłowych do obiektu docelowego. Zwraca obiekt docelowy.

```
const obj1 = { a: 1, b: 2, c: 3 }

const obj2 = Object.assign(c: 4, d: 5}, obj1)

console.log(obj2) // { a: 1, b: 2, c: 4, d: 5}
```

Zadanie 7

Stwórz kilka obiektów z różnymi i powtarzającymi się właściwościami i metodami.

Połącz te obiekty w używając Obiekt.assign i spread operator

Tworzenie obiektów i dziedziczenie
new Fn()

Constructor function to funkcja, która wywoływana jest ze słowem kluczowym **new**

new Fn():

1. Tworzy pusty obiekt.
2. Ustawia prototyp (**__proto__**) tego obiektu na Fn.prototype
3. Wywołuje funkcję w kontekście tego obiektu
4. Zwraca utworzony obiekt

Tworzenie obiektów i dziedziczenie

Constructor functions

```
function Cat(name, age){
    this.name = name
    this.age = age
    this.sound = “Meeeeow!”,
    this.makeSound = function(){
        console.log(this.sound)
    }
    this.speak = function(){
        console.log(‘Sorry cats can’t speak’)
    }
}
```

Tworzenie obiektów i dziedziczenie

Constructor functions

```
function Cat(name, age){
    this.name = name
    this.age = age
    this.sound = “Meeeeow!”,
}

Cat.prototype.makeSound = function(){
    console.log(this.sound)
}
Cat.prototype.speak = function(){
    console.log(‘Sorry cats can’t speak’)
}
```

Zadanie 8

Stwórz constructor function, która tworzy obiekt cat z właściwościami name, sound and metodami speak, makeSound.