Obiekt – podstawowy budulec

DANE (Właściwości)

ZACHOWANIE (Metody)

Obiekt kontroluje dostęp do składowych.

Technicznie obiekt to nieuporządkowany (kolejność nie ma znaczenia) zbiór wartości, które składają się z pary klucz (nazwa właściwości) – wartość (którą może być dowolny typ). Jeśli wartością jest funkcja to taką właściwość nazywamy metodą.

Zmienna przechowuje referencję do obiektu, a nie sam obiekt.

User.name - (odczytywanie czyli get )

User.age = 29 – (przypisanie nowej wartości czyli set)

Obiekty są dynamiczne więc możemy je modyfikowawć

Tworzenie metody w obiekcie: showAge() {//kod}

This czymś się staje dopiero w procesie wywołania, gdzieś prowadzi, jest referencją do czegoś

Konstruktor – jest to funkcja, która jest instrukcją tworzenia nowego obiektu (nowej instancji). Nazwę konstruktora piszemy wielką literą.

Tworzenie przepisu na konstruktor: const Animal = function(name, species) {this.name = name; itd…}

Utworzenie obiektu (konstruktora): const dog = new Animal(‘azor’, ‘owczarek’); // tworzymy instancje

Proces tworzenia instancji:

1. Operator new, który tworzy w połączeniu z konstruktorem nowy (pusty) obiekt
2. This od tego momentu wskazuje na ten obiekt (następuje wiązanie this z nowym obiektem)
3. Nowy obiekt zostaje połączony z prototypem funkcji konstruktora
4. Nowy obiekt jest zwracany (i przypisywany, referencja do niego, w zmiennej)

Instancja to obiekt stworzony zgodnie z wzorem w konstruktorze.

Powstały obiekt za pomocą konstruktora jest niezależnym bytem (instancją). Instancja ma dostęp do metod i właściwości, które miał konstruktor.

Prototyp jest częścią konstruktora, dostęp do niego mają wszystkie instancje. Wykorzystywany jest najczęściej, żeby nie powielać np. 100 razy tej samej funkcji w obiektach opartych na konstruktorze. Lepiej taką funkcję dodać jako prototyp, czyli żeby każda utworzona instancja mogła korzystać z funkcji ale jej w sobie nie mieć, tylko się do niej odwoływać.

Instancja nie jest powiązana wprost z konstruktorem, który ją stworzył, jest powiązana z jego prototypem.

Prototyp – składnia: Animal.prototype.addChildren = tu funkcja.

JS najpierw sprawdza, czy dana metoda jest obecna w konkretnej instancji, jeśli nie to szuka w prorotypie

Klasy w js to to samo co konstruktory tylko zapisane ładniej i częściej używane.

Wzór klasy: class Animal {

constructor(name,species)

{//tutaj kod} a tutaj dodajemy prototypy już bez słowa function tylko po prostu eat()}. Prototype jest zawsze aktualny.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Prototyp w klasie:

Obraz zawierający tekst

Opis wygenerowany automatycznie

W klasie możemy bezpośrednio dodać parametry dla prototypu, a nie jak w konstruktorze, gdzie trzeba wyjść z konstruktora, żeby stworzyć prototyp.

Tworzenie instancji w class wygląda tak samo jak za pomocą konstruktora. Klasa jest funkcją.

This w .addEventListener() będzie odwoływało się do przycisku, czyli do obiektu, na którym wywołuje się funkcja. Wiązanie this z obiektem następuje w chwili wywołania funkcji, a nie w chwili tworzenia funkcji. This w arrow function będzie odnosiło się do obiektu globalnego, bo funkcja strzałkowa nie tworzy własnego wiązania tylko go przejmuje z wyższego zakresu (np. przy .addEventLiustener() który jest w obiekcie globalnym, to this odniesie się do obiektu globalnego)

Hermetyzacja(ENCAPSULATION) – grupowanie (zamykanie) metod i właściwości w obiekcie.

Implementacja to sposób w jaki działa obiekt (np. silnik w samochodzie) jest ukryta przed innymi obiektami, a interfejs to to co widzi użytkownik (np. kierownica) albo implementacja jako elektrownia, a interfejs jako gniazdko.

Przykład hermetyzacji: Math.floor() nie mamy dostępu do implementacji. Interfejsem dla obiektu math jest metoda obiektu w tym wypadku floor.

Dziedziczenie(INHERITANCE) – dzięki dziedziczeniu redukujemy ilość powtarzanego kodu.

Stworzenie relacji między obiektami i grupowanie obiektów.

Zmniejszenie ilości potencjalnych błędów.

Przykład: class Dog extends Animal {} – docelowo to co ma klasa Animal plus właściwości i metody dodane w klasie Dog. Dog jest podklasą(subclass) klasyAnimal(która jest klasą nadrzędną, superclass)

Super() – służy do rozszerzania konstruktora nowej klasy o konstruktor klasy, z której dziedziczy, wykonuje się gdy używamy extends, jest odpowiedzialny za przekazanie klasie która dziedziczy tego co posiada klasa nadrzędna. Właściwości dodatkowe dla klasy podrzędnej dodajemy po super(). Jeśli nie chcemy dodawać dodatkowych klas to nie trzeba też pisać constructor i super bo to się dodaje automatycznie.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Prototype chain polega na tym, że jak mamy obiekt stworzony na bazie klasy, to ta klasa ma dostęp do prototypów z klas nadrzędnych,ostatnią klasą jest Object. Odwołując się do prototypu który jest np. w klasie Object przechodzimy przez prototype chain aby się tam dostać. Na końcu prototype chain jest null.

Polimorfizm – obiekt zachowuje się inaczej w zależności od dostarczonych danych.

Przejawy polimorfizmu:

-Implementacja metod, które zachowują się inaczej w zależności od tego jakie/ile argumentów otrzymają.

- Różna implementacja tej samej metody w różnych obiektach.

Polimorfizm polega na używaniu pewnej części kodu wiele razy.

Polimorfizm można uzyskać też wykorzystując łańcuch dziedziczenia(prototyp). Metodę jednego obiektu o tej samej nazwie przysłania metoda innego obiektu (nadrzędnego). Czyli jeśli mamy 3 metody o tej samej nazwie ale w innych obiektach które mają ze sobą więź to wykonywana jest ta która jest najbardziej zagnieżdżona, jeśli jej tam nie ma to wykonywana jest kolejna, jeśli jej tam nie ma to wykonywana jest na samym końcu ta z obiektu najbardziej nadrzędnego która ma tę metodę.

Mechanizm przeciążenia w JS pozwala budować jedną funkcję której wewnętrzna implementacja rozpoznaje jakie argumenty zostały przekazane i funkcja ta inaczej się zachowa w zależności od dodanych argumentów.

Drugim sposobem na uzyskanie przeciążenia jest tworzenie wielu funkcji o tej samej nazwie w różnych obiektach, w których zachodzi dziedziczenie.

Abstrakcja(ABSTRACTION) – model rzeczywistości, który upraszcza złożoność i pozwala przedstawić problem (zadanie) za pomocą obiektów i relacji między nimi.

Abstrakcja określa jakie cechy musi posiadać i jakie zadanie realizować obiekt.

Abstrakcja to umiejętność pozwalająca na modelowanie programu/projektu.

Abstrakcja określa relacje między obiektami.

Przykładowe relacje między obiektami:

Kompozycja – obiekt zawierają inne obiekty. Kompozycja zakłada, że jeden obiekt jest wbudowany w inny obiekt. Przy czym obiekt wbudowany jest zależny od obiektu, w którym się znajduje, a jego istnienie poza obiektem, w który jest wbudowany nie ma sensu. (Przykład: obiekt komin w obiekcie elektrownia)

Asocjacja – każdy obiekt tej relacji istnieje niezależnie. Pomimo to istnieje relacja (powiązanie) między obiektami. Istnienie jednego obiektu nie jest potrzebne do istnienia drugiego. (przykład: relacja wujek- ciocia)

Agregacja – obiekt składa się z innych obiektów (agreguje je). Sens istnienia agregatu (obiektu głównego) polega na posiadaniu obiektów ,które przechowuje (choć te obiekty mogą istnieć poza nim). (przykład: drużyna jako obiekt główny i piłkarze jako obiekty przechowywane w obiekcie głównym. Drużyna bez piłkarzy nie ma sensu).

Sprawdzanie czy jakiś obiekt jest instancją danego konstruktora: arr instanceof Array, tutaj to będzie zapytanie, czy obiekt arr jest instancją konstruktora Array, zwraca true lub false.

Object.getPrototypeOf(arr) – zwraca prototyp do którego dostęp ma wskazana w nawiasie instancja.

Static oznacza metodę statyczną która jest dostępna tylko z poziomu klasy.Czyli jeśli chcemy, żeby jakaś metoda była dostępna tylko z poziomu klasy, a nie też instancji to używamy static a później piszemy metodę.Przydatne również, gdy chcemy skorzystać z metody jakiejś klasy bez konieczności tworzenia instancji.

Klasy nie podlegają hoistingowi. Klasy są również bezpieczniejsze niż funkcje, ponieważ nie da się ich wywołać bez ‘new’ przed nazwą klasy. This wywołany w funkcji dotyczy obiektu Window, a this wywołany w klasie dotyczy tej klasy.

Tryb ścisły w js pomaga tworzyć lepszy kod, możemy ten tryb zdefiniować albo dla funkcji albo dla całego pliku(najczęściej). Aby włączyć tryb ścisły piszemy na początku pliku ‘use strict’. Od ES6 mamy te mechanizmy już wbudowane w wielu elementach.

This może przyjmować różne wartości w zależności gdzie i w jaki sposób funkcja jest wywoływana. Jeśli jest wywoływana od razu np. fun() a nie np. przez obj.fun() to będzie próba przypisania do obiektu window, a jeśli użyjemy obj.fun() to przypiszemy this do obj więc nie będzie błędu.

This – słowo zawierające odwołanie do obiektu. JS przypisuje zgodnie z regułami do ‘this’ wiązanie z obiektem.

O tym czym jest this decyduje źródło (sposób) wywołania funkcji.

Przypisanie do obiektu z this jest zawsze w chwili wywołania funkcji. Domyślnie ma charakter niejawny (a więc nie musimy wskazywać czym jest this, to stanie się automatycznie)

This tworzy kontekst (obiekt) do którego możemy się odwołać w kodzie.

Dlaczego this jest potrzebne:

- nie ma potrzeby używania precyzyjnych funkcji osobnych dla każdego obiektu

- możliwość wielokrotnego użycia tej samej funkcji.

- czystość w kodzie, łatwiejsza edycja kodu.

Kiedy wiązanie this powstaje:

- wiązanie this powstaje w chwili wykonywania metody i nie jest zależne wprost od miejsca deklaracji funkcji, a od sposobu jej wyowłania.

- w chwili wykonywania funkcji, powstaje konstekst wykonawczy, którego częścią jest this.

Reasumując: Źródłem powstania this jest wywołanie funkcji. Wiązanie powstaje w chwili wywołania funkcji/metody. Wiązanie z this powstaje nawet jeśli w funkcji nie wykorzystujemy this.

Reguły this:

- wiązanie domyślne this to obiekt globalny, chyba że używamy ‘use strict’ (tryb ścisły), które na to nie pozwoli(będzie undefined).

-wiązanie niejawne np. user.fun() teraz this będzie userem

-wiązanie jawne – możemy powiedzieć funkcji z czym ma powiązać this. Mówimy to przy wywołaniu funkcji. Metody call i apply.

Obj.Call(first) – ustawia this na pierwszy argument w nawiasie, tutaj first, i wywołuje tutaj Obj. Drugi i kolejne argumenty to argumenty funkcji którą wywołaliśmy tutaj argumenty funkcji Obj. Jeśli funkcja nie ma argumentów to zostawiamy tylko drugi i trzeci argument call() puste.

Obj.apply() – to to samo ale jako drugi argument wpisuje się tablicę

-wiązanie twarde: bind – nowa funkcja ze stałym this. Ta funkcja nie wywołuje się jak apply(), tylko przypisuje do pierwszego argumentu w nawiasie objekt do którego odwoływać będzie this. Trzeba ją gdzieś przypisać, żeby jej użyć, bo się sama nie wywołuje.

-wiązanie z operatorem new

Arrow function a this: Arrow function nie tworzy podczas wywołania własnego wiązania this, a je dziedziczy, co czasami jest plusem, a czasami minusem.

Problem utraty wiązania this:

Jeśli wywołujemy obj.fun() i w fun() jest kolejna funkcja to this w tej kolejnej funkcji nie będzie wskazywał na obj tylko na window, bo utracił wiązanie.

Rozwiązanie1: przypisanie this do innej zmiennej i użycie tej zmiennej w funkcji która straciła wiązanie.

Rozwiązanie2: bez funkcji – za pomocą pętli for of pętla nie tworzy nam nowego this, więc nie ma problemu z utratą wiązania. For(const child of this .children)

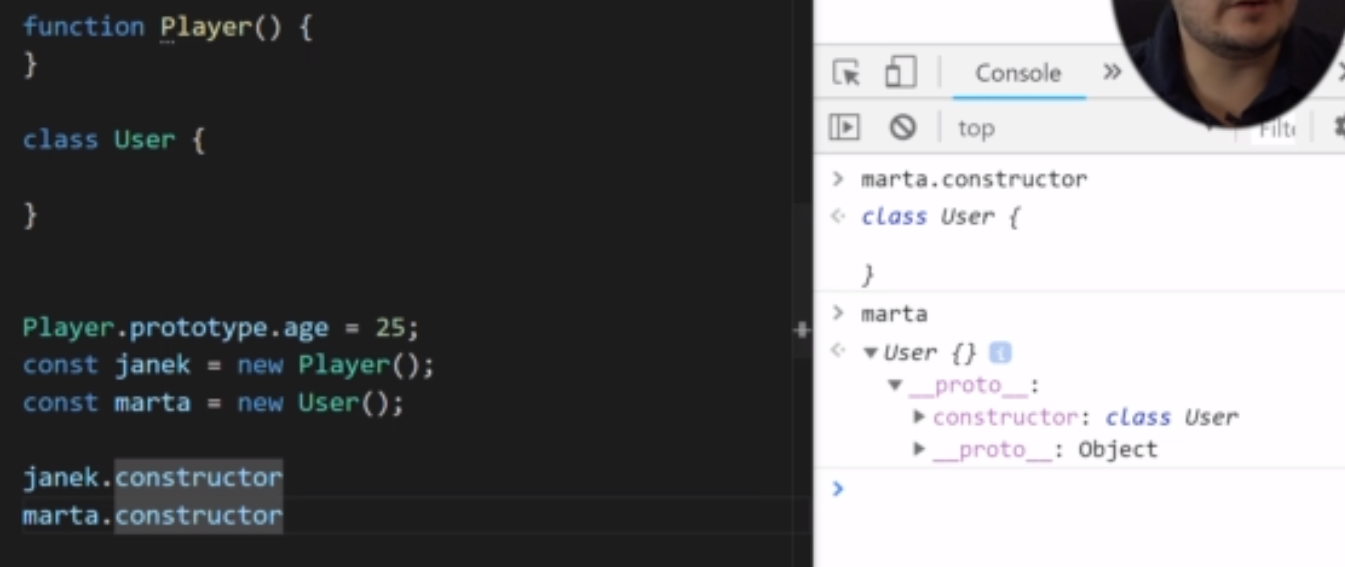
Rozwiązanie3: arrow function – w forEach() zamiast function() używamy () => funkcja strzałkowa nie tworzy swojego this tylko go dziedziczy od rodzica, więc też nie ma problemu z zerwaniem wiązania.

Rozwiązanie4: użycie bind(). Fun(…).bind(this) teraz ta funkcja nie utraci this tylko dostanie this który się odwołuje do obiektu. Zastępuję funkcję i daje jej this obiektu w którym została wykonana funkcja.

Zmienna ukryta: let \_zmienna, this.wywolanie = () => \_zmienna; instancja.wywolanie()

Tworzenie obiektu błędu: throw New Error(‘wiadomosc’) - tworzy w konsoli errora, wygląda jak te errory gdy coś zwaliłeś

Konstruktor obiektu jest zapisywany w prototypie instancji:



Metody danych obiektów np. map, filter, find w obiekcie Array są dostępne właśnie dzięki prototypowi:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst, urządzenie, znak, pomiar

Opis wygenerowany automatycznie

Zmienna instanceof Array => sprawdza czy zmienna jest instancją Array, Object itp.

Prototyp jest obiektem.

Instancja nie jest połączona bezpośrednio z obiektem, który ją stworzył, a z prototypem tego obiektu.

Constructor(zmienna1, …zmienne) – akceptuje dowolną liczbę parametrów, tak jak w funkcji.

Jak zrobić metodę, która jest dostępna tylko odwołując się do klasy?

Używając static przed nazwą metody.

Dzięki temu do takiej funkcji nie możemy odwołać się podając nazwę instancji. Jedynie podając nazwę klasy i po kropce nazwę metody przed którą było static. Np. tutaj Family.makeFamily()

Obraz zawierający tekst

Opis wygenerowany automatycznie

Class Animl {} – nie podlega hoistingowi, przypisanie do zmiennej klasy bez new spowoduje błąd

Function Animal() {} – podlega hoistingowi, przypisanie do zmiennej klasy bez new spowoduje wywołanie się funkcji(mniej bezpieczne niż class)

Cechy wspólne - nie można przypisać do zmiennej bez „new” przed nazwą klasy

Dziedziczenie

Const Mammal extends Animal – klasa Mammal dziedziczy klasę Animal

Super(zmienne które chcemy przekazać do konstruktora nadrzędnej klasy) – wywołuje konstruktor klasy, od której dziedziczymy(nadrzędnej)

W konstruktorze najpierw mamy metodę super() a dopiero później możemy tworzyć zmienne tej konkretnej klasy.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeśli nie podamy constructor() w klasie, która dziedziczy to odziedziczy ona wszystko automatycznie. Constructor() i super() i tak się wywoła automatycznie. Ale jeśli zrobimy własny konstruktor i nie dodamy super to się już automatycznie super() nie doda i wyskoczy błąd jak będziemy chcieli użyć.

THIS

Tryb ścisły ‘use strict’ nie pozwoli, żeby this się domyślnie wiązało z obiektem window w funkcji.

This – słowo zawierające odwołanie do obiektu. JS przypisuje do ‘this’ wiązanie z obiektem.

O tym czym jest ‘this’ decyduje sposób wywołania funkcji.

Przypisanie do obiektu w this jest zawsze w chwili wywołania funkcji.

Domyślnie ma charakter niejawny, więc nie musimy wskazywać czym jest this, to stanie się automatycznie

This tworzy kontekst (obiekt) do którego możemy się odwołać w kodzie.

Co daje this?

Nie ma potrzeby używania precyzyjnych funkcji(osobnych dla każdego obiektu). Możliwość wielokrotnego użycia tej samej funkcji.

Czystość w kodzie, łatwiejsza edycja kodu.

Kiedy powstaje this?

Wiązanie this powstaje w chwili wykonania metody i nie jest zależne wprost od miejsca deklaracji funkcji a od sposobu jej wywołania.

W chwili wykonania funkcji powstaje kontekst wykonawczy, którego częścią jest this.

Jeśli przypisujesz obiekt do jakiejś zmiennej to nie tworzysz kopii tylko referencję do obiektu w tej zmiennej.

Reguły this:

1. Wiązanie domyślne this to obiekt globalny, chyba że używamy ‘use strict’ (tryb ścisły), które na to nie pozwoli (będzie undefined)
2. Wiązanie niejawne – co jest obiektem kontekstu, co jest źródłem wywołania
3. Możemy powiedzieć funkcji z czym ma powiązać this. Mówimy to przy wywołaniu funkcji. Metody call i apply obie niemal identyczne.

Funkcja.call(to co ma być thisem, parametry funkcji)

1. Wiązanie twarde – bind – nowa funkcja ze stałym this.

Funkcja.bind(co ma być thisem) – tutaj funkcja się nie wywoła w przeciwieństwie do call()

Takie przypisanie trzeba gdzieś przypisać do zmiennej i dopiero zmienną można już wywoływać, gdzie się chce.

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Wiązanie z operatorem new. Tutaj thisem jest Stasiek.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Arrow function nie tworzy podczas wywołania własnego wiązania this, a je dziedziczy.

Problem utraty wiązania

1. Gdy mamy funkcję w funkcji i ta druga funkcja ma wiązanie domyślne, czyli this jest obiektem window, chyba że zastosujesz arrow function.
2. Można też przypisać thisa do zmiennej that i użyć tej zmiennej w drugiej funkcji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Bez funkcji – za pomocą pętli for of, bo pętla nie tworzy nowego kontekstu, wiązania.

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Bind

Obraz zawierający tekst

Opis wygenerowany automatycznie

5. Array.forEach( callback, this\_object )

1. const szarik6= {
2. children: ['fafik', 'żaba'],
3. showChildren: function() {
4. this.children.forEach(function(child, index) {
5. console.log(this.children[index]);
6. }, this)
7. // metoda forEach może przyjąć dwa argumenty - callback i referencję na obiekt która w funkcji callback wiązana jest do this
8. },
9. }
10. szarik6.showChildren()

Prywatne i publiczne właściwości

Domyślnie wszystkie metody i właściwości obiektów są publiczne w JS.

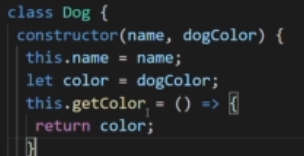
2 sposoby na stworzenie właściwości prywatnej:

1. Konwencja – czyli umawiamy się, że nie używamy bezpośrednio właściwości z podkreślnikiem i zamiast tego tworzymy prosty interfejs w postaci metody który obsługuję tę właściwość

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. Wykorzystanie domknięć – czyli tworzysz normalnie zmienną z użyciem let/const tylko po wywołaniu instancji taka zmienna przestaje istnieć i żeby ją jakoś zachować to trzeba utworzyć funkcję, która zwraca tę zmnienną z let/const i ona sobie w funkcji żyje i nie znika po wywołaniu instancji.



Moduły w js – przydatny link:

https://www.freecodecamp.org/news/javascript-modules-explained-with-examples/#:~:text=A%20module%20in%20JavaScript%20is,object%20accessible%20to%20other%20modules.