hillel IT school

Lesson 13

План заняття

- ООП;
- Class constructors;
- Геттери та Сеттери в об'єктах та класах;
- Робота з class;
- Спадкування в classes;
- Використання батьківських методів та конструктора;
- Статичні властивості та методи в classes.

Class

Клас - це опис того, якими властивостями і поведінкою матиме об'єкт. А об'єкт — це екземпляр із власним станом цих властивостей.

"Властивості" — це звичайні змінні, просто вони є атрибутами якогось об'єкта (їх називають полями об'єкта).

"Поведінка" - це функції об'єкта (їх називають методами), які теж є атрибутами об'єкта. Різниця між методом об'єкта та звичайною функцією лише в тому, що метод має доступ до власного стану через поля.

Клас у програмі – це визначення «типу» кастомної структури даних, куди входять як дані, і застосовується щодо них поведінка. Класи визначають роботу таких структур даних, але самі при цьому конкретними значеннями не є. Для отримання значення, яке можна буде використовувати в програмі, клас потрібно інстанціювати (за допомогою ключового слова new) один або більше разів.

Class

- Пам'ятайте, що класи не є фактичними сутностями або об'єктами, а являють собою схеми, які ми використовуємо для їх створення.
- За згодою імена класів оголошуються з першої великої літери та прописуються в CamelCase. Ключове слово class створює константу, виключаючи можливість її подальшого перевизначення.
- Класи завжди повинні містити метод-конструктор, який служить майбутньому інстанцію самого класу. У JS конструктор - це проста функція, що повертає об'єкт. Єдина особливість тут у тому, що при виклику з ключовим словом new вона надає свій прототип у вигляді прототипу об'єкта, що повертається.
- Ключове слово this вказує на сам клас і служить для визначення його властивостей усередині конструктора.
- Методи можна додавати простим визначенням імені функції та її виконуваного коду.
- JS - це мова, заснована на прототипах, і всередині нього класи використовуються тільки як синтаксичний цукор.

class in JS

Конструкція class User {...} в дійсності робить наступне:

- Створює функцію, що називається User та стає результатом оголошення класу. Код цієї функції береться з методу constructor (вважається порожнім, якщо ми не написали такий метод).
- Записує методи класу до User.prototype.

Важливою відмінністю полів класу є те, що вони задаються в окремих об’єктах, а не в User.prototype

class with f.constructors

Існують важливі відмінності:

1. Функція, що створена за допомогою class, позначена спеціальною внутрішньою властивістю [[IsClassConstructor]]: true. Так що це не зовсім те саме, що створити її вручну.
2. Методи класу неперелічувані. Оголошення класу встановлює прапор enumerable у false для всіх методів в "prototype".
3. Клас завжди use strict. Весь код всередині конструкції класу автоматично знаходиться в суворому режимі.
4. Додає спеціальніможливості.

Class expression

Так само, як функції, класи можуть бути визначені всередині іншого виразу, передані, повернуті, присвоєні тощо

get/set

Подібно до літералів об’єктів, класи можуть включати геттери/сеттери, обчислені атрибути тощо.

Методи класу

Функції в JavaScript мають динамічний this. Він залежить від контексту виклику. Якщо метод об’єкта передається і викликається в іншому контексті, this більше не буде посиланням на цей об’єкт.

Існує два підходи до розв’язання проблеми з втратою this:

- Передати функцію-обгортку, наприклад setTimeout(() => button.click(), 1000).
- Зв’язати метод з об’єктом за допомогою функції bind, наприклад у конструкторі.

Наслідування класу

Наслідування класу – це коли один клас розширює інший.

Таким чином, ми можемо створити нову функціональність на основі тої, що існує.

Синтаксис, щоб розширити інший клас: class Child extends Parent.

Внутрішньо, ключове слово extends працює за допомогою механіки прототипу. Він встановлює в Child.prototype.[[Prototype]] значення Parent.prototype. Тому, якщо метод не знайдено в Child.prototype, JavaScript бере його з Parent.prototype.

Синтаксис класу дозволяє вказати не лише клас, але будь-який вираз після extends.

Успадкування

Успадкування– це можливість створювати класи з урахуванням інших класів. За допомогою цього принципу можна визначати батьківський клас (з потрібними властивостями та методами), а потім дочірній клас, який успадковуватиме від батька всі властивості та методи.

- Клас може успадковувати лише від одного з батьків. Розширювати кілька класів не можна, хоча цього є свої хитрощі.
- Ви можете безмежно збільшувати ланцюжок успадкування, встановлюючи батьківський, «дідівський», «прадідівський» тощо класи.
- Якщо дочірній клас успадковує якісь властивості від батьківського, він спочатку повинен привласнити ці властивості через виклик функції super() і лише потім встановлювати свої.
- При успадкування всі батьківські методи та властивості переходять до нащадка. Тут ми не можемо вибирати, що саме успадковувати (так само, як не можемо вибирати переваги або недоліки, які ми отримуємо від батьків при народженні. До теми вибору ми ще повернемося при розгляді композиції).
- Дочірні класи можуть перевизначати батьківські властивості та методи.

Поліморфізм

Поліморфізм є одним із принципів об'єктно-орієнтованого програмування (ООП). Це допомагає проектувати об'єкти таким чином, щоб вони могли спільно використовувати або перевизначати будь-яку поведінку з наданими конкретними об'єктами.

Саме слово означає багато форм. Існує багато тлумачень того, що саме воно означає, але ідея полягає в здатності викликати той самий метод для різних об'єктів, і при цьому кожен об'єкт реагує по-своєму.

Щоб це сталося, поліморфізм використовує успадкування.

Перевизначення методу

Щоб росширити функціонал батьківського класу треба використовувати ключове слово super()

- super.method(...), щоб викликати батьківський метод.
- super(...), щоб викликати батьківський конструктор (лише в нашому конструкторі).

Пам’ятайте!

Стрілкові функції не мають super! Якщо super доступний, то він береться із зовнішньої функції.

Перевизначення конструктора

Відповідно до специфікації, якщо клас розширює ще один клас і не має конструктора, то автоматично створюється “порожній” конструктор.

```
class Child extends Parent{
  // генерується для класів-нащадків без власних конструкторів
  constructor(...args) {
    super(...args);
  }
}
```

Перевизначення конструктора

Конструктори в класі, що наслідується, повинні викликати super(...) і (!) зробити це перед використанням this.

У JavaScript існує відмінність між функцією-конструктором класу, що успадковується (так званого “похідного конструктора”), та іншими функціями. Похідний конструктор має особливу внутрішню власність [[ConstructorKind]]:"derived". Це особлива внутрішня позначка.

Ця позначка впливає на поведінку функції-конструктора з new.

- Коли звичайна функція виконується з ключовим словом new, воно створює порожній об’єкт і присвоює його this.
- Але коли працює похідний конструктор, він не робить цього. Він очікує, що батьківський конструктор виконує цю роботу.

Таким чином, похідний конструктор повинен викликати super, щоб виконати його батьківський (базовий) конструктор, інакше об’єкт для this не буде створено. І ми отримаємо помилку.

Проблеми успадкування

Подивимось у код:

1. Всередині kusKus.run(), рядок (**) викликає wolf.run надаючи йому this=wolf.
2. Тоді в рядку (*) в wolf.eat, ми хотіли б передати виклик ще вище в ланцюгу наслідування, але this=kusKus, тому this.__proto__.run знову wolf.run!
3. Отже, wolf.eat викликає себе в нескінченній петлі, тому що він не може піднятися вище.

Проблема не може бути вирішена лише за допомогою this.

Тому у JS шснує символ [[HomeObject]]

[[HomeObject]]

Щоб забезпечити ланцюг успаадкування, JavaScript додає ще одну спеціальну внутрішню власність для функцій: [[HomeObject]].

Коли функція вказана як метод класу або об’єкта, її властивість [[HomeObject]] стає цим об’єктом.

Тоді super використовує цю властивість для знаходження батьківського прототипу та його методів.

Як ми знаємо раніше, взагалі функції “вільні”, тобто не пов’язані з об’єктами в JavaScript. Таким чином, їх можна скопіювати між об’єктами та викликати з іншим – this.

Саме існування [[HomeObject]] порушує цей принцип, оскільки методи запам’ятовують їх об’єкти. [[HomeObject]] не можна змінити, тому цей зв’язок назавжди.

Єдине місце в мові, де [[HomeObject]] використовується – це super. Отже, якщо метод не використовує super, то ми можемо все одно враховувати його вільним та копіювати між об’єктами. Але з super речі можуть піти не так.

Статичні властивості та методи

Ми також можемо присвоїти метод самій функції класу. Такі методи називаються статичними.

Зазвичай статичні методи використовуються для реалізації функцій, які належать до класу, але не до будь-якого окремого його об’єкта.

Значенням this при виклику User.greeting() є сам конструктор класу User (правило «об’єкт перед крапкою»).

Статичні методи також використовуються в класах, що пов’язані з базою даних, для пошуку/збереження/видалення записів із бази даних.

Важливо!

Статичні методи викликаються для класів, а не для окремих об’єктів.

Статичні властивості та методи

Статичні властивості також можливі, вони виглядають як звичайні властивості класу, але до них додається static

Статичні властивості та методи

Статичні властивості та методи наслідуються.

Як це працює? Знову ж таки, використовуючи прототипи. Як ви вже могли здогадатися, extends дає UsersApi посилання [[Prototype]] на API

Отже, UsersApi extends API створює два посилання [[Prototype]]:

- Функція UsersApi прототипно успадковує від функції API.
- UsersApi.prototype прототипно успадковує від API.prototype.

Дякую за увагу