

Lesson 14

План заняття

- Інтерфейси
- Абстракції
- Абстрактні класи
- Інкапсуляція
- Приватні властивості класу
- Міксини

Інтерфейс

Інтерфейс - це сукупність методів та правил взаємодії елементів системи. Іншими словами, інтерфейс визначає, як елементи будуть взаємодіяти між собою.

- Інтерфейс дверей – наявність ручки;
- Інтерфейс автомобіля - наявність керма, педалей, важеля коробки;
- Інтерфейс дискового телефону - трубка + дисковий набір номера.

Коли ви використовуєте ці "об'єкти", ви впевнені, що ви зможете використовувати їх подібним чином. Завдяки тому, що ви знайомі з їхнім інтерфейсом.

"Інтерфейс визначає як ми можемо використовувати об'єкт"

Головна відмінність класу від інтерфейсу - у тому, що клас складається з інтерфейсу та реалізації.

Будь-який клас завжди явно оголошує свій інтерфейс - те, що є при використанні класу ззовні.

Як такого у JS не існує інтерфейсів, тобто ви не можете явно типізувати та описати сутність.

Абстракція

Абстракція - це спосіб створення простої моделі, яка містить лише важливі властивості з погляду контексту додатку, з більш складної моделі. Іншими словами - це спосіб приховати деталі реалізації та показати користувачам лише функціональність. Абстракція ігнорує нерелевантні деталі та показує лише необхідні. Важливо пам'ятати, що ми можемо створити екземпляр абстрактного класу.

Все програмне забезпечення – це абстракція, що приховує всю важку роботу та бездумні деталі.

Інтерфейс & Абстрактний клас

У JS не існує абстрактних класів.

Абстрактним називається такий клас, екземпляр якого не можна створити сам собою – він служить основою для інших класів.

Наприклад, геометрична фігура може представляти коло, квадрат, трикутник, але як такої геометричної фігури самої по собі не існує.

У абстрактних класах можуть існувати абстрактні методи та властивості.

Typescript example:

```
abstract class Figure {  
    abstract x: number;  
    abstract y: number;  
    abstract getArea(): void;  
}
```

Інтерфейс & Абстрактний клас

1. Інтерфейс описує лише поведінку. Він не має стану. А в абстрактного класу стан є: він описує і те, й інше.
2. Абстрактний клас пов'язує між собою та поєднує класи, що мають дуже близький зв'язок. У той самий час, той самий інтерфейс можуть реалізувати класи, які взагалі немає нічого спільного.
3. Класи можуть реалізовувати скільки завгодно інтерфейсів, але успадковуватися можна лише від одного класу.

Інтерфейс

У об'єктно-орієнтованому програмуванні, властивості та методи розділені на дві групи:

- Внутрішній інтерфейс – методи та властивості, доступні в інших методах класу, але не ззовні.
- Зовнішній інтерфейс – методи та властивості, доступні також ззовні класу.

У JavaScript існує два типи полів об'єктів (властивостей та методів):

- Публічні: доступні з будь-якого місця. Вони складають зовнішній інтерфейс. Досі ми використовували лише публічні властивості та методи.
- Приватні: доступний тільки зсередини класу. Вони для внутрішнього інтерфейсу.

Захищені поля не реалізовані в JavaScript на рівні мови, але на практиці вони дуже зручні, тому вони емулюються.

Інкапсуляція

Інкапсуляція – це принцип, згідно з яким будь-який клас або будь-яка частина системи повинна розглядатися як «чорна скринька»: користувач класу або підсистеми повинен бачити лише інтерфейс (тобто список декларованих властивостей та методів) і не вникати у внутрішню реалізацію.

Іншими словами це означає здатність об'єкта «вирішувати», яку інформацію він розкриватиме для зовнішнього світу, а яку ні. Реалізується цей принцип через громадські та закриті властивості та методи.

У JS усі властивості об'єктів та методи за замовчуванням є публічними. «Публічне» означає можливість доступу до властивості/методу об'єкта ззовні його тіла

Інкапсуляція корисна у випадках, коли нам потрібні певні властивості або методи виключно для внутрішніх процесів об'єкта, і ми не хочемо розкривати їх зовні.

З точки зору ООП, відокремлення внутрішнього інтерфейсу від зовнішнього називається інкапсуляція.

Емуляція

Захищені властивості зазвичай починають з підкресленням `_`. Це не синтаксис на рівні мови, а лише відома домовленість між програмістами, що такі властивості та методи не повинні бути доступними ззовні.

Щоб контролювати змінні використовуйте `get/set`

Захищені поля успадковуються!

Приватна властивість

Приватні властивості і методи повинні починатися з #. Вони доступні лише з класу.

На рівні мови, # – це особливий знак того, що поле є приватним. Ми не можемо отримати доступ до нього ззовні або з наслідуваних класів.

Приватні поля не конфліктують з публічним. Ми можемо мати як приватне #salary та і публічне salary поле одночасно.

На відміну від захищених, приватні поля забезпечуються самою мовою.

Розширення вбудованих класів

Вбудовані класи, такі як Array, Map та інші, також розширюються.

Пам'ятайте!

Вбудовані класи не успадковують статистику один від одного.

Перевірка класу: "instanceof"

Оператор instanceof дозволяє перевірити, чи належить об'єкт до певного класу. Він також враховує наслідування.

Алгоритм операції `obj instanceof Class` працює приблизно наступним чином:

1. Якщо є статичний метод `Symbol.hasInstance`, тоді він просто викликається: `Class[Symbol.hasInstance](obj)`. Він повинен повернути `true` або `false`, ось і все. Ось як ми можемо задати поведінку `instanceof`.
2. Більшість класів не мають `Symbol.hasInstance`. У цьому випадку використовується стандартна логіка: `obj instanceof Class` перевіряє чи `Class.prototype` дорівнює одному з прототипів у ланцюжку прототипів `obj`.

.toString для визначення типу

Можна зробити метод `toString` набагато потужнішим. Ми можемо використовувати його як розширений `typeof` і альтернативу `instanceof`.

Нам потрібно викликати метод `toString()` зновим контекстом наприклад `this=arr`. Всередині алгоритм `toString` перевіряє `this` і повертає відповідний результат.

Також поведінку методу об'єкта `toString` можна налаштувати за допомогою спеціальної властивості `Symbol.toStringTag`. Для більшості специфічних для середовища об'єктів така властивість є, наприклад `window`, `document`, `XMLHttpRequest` ...

Міксини

В JavaScript ми можемо успадкуватися лише від одного об'єкта. Може бути лише один `[[Prototype]]` для об'єкта. І клас може розширювати лише один інший клас.

Як визначено у Вікіпедії, `mixin` – це клас, що містить методи, які можуть бути використані іншими класами без необхідності успадкуватися від нього.

Іншими словами, міксин забезпечує методи, які реалізують певну поведінку, але ми не використовуємо його самостійно, ми використовуємо його, щоб додати цю поведінку до інших класів.

Дякую за увагу