متدهای object و this

objectها معمولا ساخته می‌شوند تا موجودیت‌های دنیای واقعی را به نمایش بگذارند مانند کاربران ، سفارشات و ...

```
let user = {
  name: "John",
  age: 30
};
```

در دنیای واقعی یک کاربر می‌تواند کارهایی (act) از قبیل انتخاب کالا ، ورود ، خروج و غیره را انجام دهد .

کارها (actions) در جاوا اسکریپت می توانند توسط فانکشن‌هایی در propertyها نمایش داده شوند .

مثال‌هایی از method

برای شروع ، می‌خواهیم به کاربر یاد بدهیم که سلام کند :

```
let user = {
  name: "John",
  age: 30
};

user.sayHi = function() {
  alert("Hello!");
};

user.sayHi(); // Hello!
```


در اینجا ما از function expression استفاده کردیم تا یک فانکشن بسازیم و آن را به object property کاربر با نام sayHi اختصاص داده‌ایم . حال می‌توانیم بوسیله user.sayHi() این فانکشن را صدا بزنیم . الآن کاربر می‌تواند حرف بزند!

فانکشنی که یک propery از یک object باشد ، method نامیده می‌شود .

پس در اینجا sayHi یک method از آبجکت user است .

هم چنین ، از فانکشن‌های از پیش تعریف شده نیز می‌توانیم به عنوان method استفاده کنیم . مانند :

```
let user = {
  // ...
};

// first, declare
function sayHi() {
  alert("Hello!");
};

// then add as a method
user.sayHi = sayHi;

user.sayHi(); // Hello!
```

برنامه نویسی شی گرا (Object-oriented)

وقتی از objectها برای نمایش موجودیت‌ها استفاده می‌کنیم ، در واقع داریم از object-oriented programming استفاده می‌کنیم . (به صورت اختصار به آن OOP می‌گویند .)

OOP مفهوم بزرگی است و دانش مورد نیاز خود را می‌طلبد . چگونه موجودیت‌ها را به درستی انتخاب کنیم؟ چگونه تعامل بین موجودیت‌ها را ساماندهی کنیم؟ به جواب این سوالات ، معماری (architecture) می‌گویند و کتابهای خوبی در این زمینه وجود دارد مانند :

“Design Patterns: Elements of Reusable Object-Oriented Software” by E. Gamma, R. Helm, R. Johnson, J. Vissides
“Object-Oriented Analysis and Design with Applications” by G. Booch

---

راه کوتاهتر برای نوشتن متدها

برای نوشتن methodهای یک object ، syntax کوتاهتری نیز وجود دارد :

```
user = {
  sayHi: function() {
    alert("Hello");
  }
};
```

```
// method shorthand looks better, right?
user = {
  sayHi() { // same as "sayHi: function(){...}"
    alert("Hello");
  }
};
```

همانطور که مشاهده کردید ، می‌توانیم کلمه "function" را حذف کرده و فقط sayHi() را بنویسیم .

اگر بخواهیم رو راست باشیم ، این دو روش کاملا یکسان نیستند . تفاوت‌هایی ریزی وجود دارد که به ارث بری objectها مربوط می‌شود (در ادامه توضیح داده خواهد شد) ، ولی برای الآن می‌توانیم این دو روش را یکسان در نظر بگیریم . در اکثر مواقع ، روش دوم ارجحیت دارد .

this در methodها

معمولا methodهای یک object برای انجام دادن کارهایشان نیاز دارند تا به اطلاعات ذخیره شده در آن object دسترسی داشته باشند .

برای مثال ، کد داخل user.sayHi() ممکن است به نام user نیاز داشته باشد .

برای دسترسی به یک object ، یک method می‌تواند از کلمه this استفاده نماید .

مقدار this ، همان object قبل از dot (نقطه) است ، یعنی همان objectای که از آن برای فراخوانی method استفاده نموده‌ایم .

برای مثال :

```
let user = {
  name: "John",
  age: 30,
  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }
};
user.sayHi(); // John
```

در مثال فوق ، مقدار this در حین اجرای user.sayHi() ، آبجکت user است .

از لحاظ فنی ، بدون this و بوسیله متغیر بیرونی (outer variable) هم می‌توان به object دسترسی داشت :

```
let user = {
  name: "John",
  age: 30,
  sayHi() {
    alert(user.name); // "user" instead of "this"
  }
};
```


ولی چنین کدی قابل اعتماد نیست . اگر تصمیم بگیریم که user را درون متغیر دیگری کپی کنیم ، مثلا admin = user و به user مقدار دیگری دهیم ، admin به object اشتباهی دسترسی خواهد داشت . مثال زیر را در نظر بگیرید :

```
let user = {
  name: "John",
  age: 30,
  sayHi() {
    alert( user.name ); // leads to an error
  }
};
let admin = user;
user = null; // overwrite to make things obvious
admin.sayHi(); // TypeError: Cannot read property 'name' of null
```

در مثال فوق ، اگر به جای user.name از this.name استفاده کنیم ، کد فوق درست کار خواهد کرد .

در جاوا اسکریپت ، کلمه this ، bound نشده است .

در جاوااسکریپت ، کلمه bound متفاوت از سایر زبان‌های برنامه نویسی عمل می‌کند و می‌تواند داخل هر فانکشنی استفاده شود حتی اگر آن فانکشن ، متدی از یک object نباشد . در مثال زیر هیچ خطای syntaxای وجود ندارد :

```
function sayHi() {
  alert( this.name );
}
```

مقدار کلمه this در زمان اجرا تعیین شده و به contextای که در آن اجرا می‌شود بستگی دارد .

در مثال زیر ، یک فانکشن را به دو object مختلف انتساب می‌دهیم و می‌بینیم که کلمه this برای هر کدام به طور مجزا رفتار می‌کند :

```
let user = { name: "John" };
let admin = { name: "Admin" };
function sayHi() {
  alert( this.name );
}
// use the same function in two objects
user.f = sayHi;
admin.f = sayHi;
// these calls have different this
// "this" inside the function is the object "before the dot"
user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)
admin['f'](); // Admin (dot or square brackets access the method – doesn't matter)
```


قانون کلی ساده است : اگر obj.f() صدا زده شود ، کلمه this برابر با objای است که فراخوانی را انجام داده است . پس در مثال فوق هم user ممکن است باشد هم admin .

فراخوانی بدون object یعنی this == undefined

در مثال فوق حتی می‌توانستیم فانکشن را بدون هیچ objectای فراخوانی کنیم :

```
function sayHi() {
  alert(this);
}
sayHi(); // undefined
```

در این حالت مقدار this برابر است با undefined (در حالت strict) . اگر بخواهیم به مقدار this.name دسترسی داشته باشیم با خطا مواجه می‌شویم .در حالت غیر strict ، در چنین مواقعی مقدار this برابر می‌شود با global object (این object در مرورگر window است که در ادامه در مورد آن صحبت خواهیم نمود .) در واقع این یک رفتار قدیمی است که بوسیله عبارت "use strict" می‌توان آن را اصلاح نمود . ولی به صورت کلی همچین فراخوانی‌هایی ، خطاهای برنامه نویسی محسوب می‌شود . اگر درون فانکشنی از this استفاده شده است ، پس انتظار می‌رود که فراخوانی آن درون یک object context انجام شود .

عواقب unbound کردن this

اگر شما از زبان برنامه نویسی دیگری آمده باشید ، احتمالا از کلمه this به صورت bound شده استفاده نموده‌اید . یعنی this در متدهایی که درون یک object تعریف می‌شوند ، همیشه برابر است با همان object .

در جاوااسکریپت this آزاد است و مقدار آن در زمان اجرا تعیین می‌شود و به جایی که method تعریف شده است بستگی ندارد بلکه به آنچه قبل از dot (نقطه) است بستگی دارد .

مفهوم محاسبه مقدار this در زمان اجرا ، معایب و محاسن خودش را دارد . از طرفی یک فانکشن می‌تواند در objectهای متفاوتی مورد استفاده قرار گیرد ، از طرف دیگر چنین انعطاف پذیری‌ای امکان بروز خطاهای متعدد را فراهم می‌آورد .

ما در جایگاهی نیستیم که طراحی جاوااسکریپت رو مورد قضاوت قرار دهیم . ما فقط ویژگی‌های آن را شناخته و از آن به نفع خودمان و برای پیشگیری از بروز خطا استفاده می‌کنیم .

Arrow Functionها ، this ندارند

Arrow functionها خاص هستند چرا که this مخصوص خودشان را ندارند . اگر درون چنین فانکشن‌هایی از this استفاده کنیم ، در واقع داریم از this فانکشن معمولی بیرونی استفاده می‌کنیم .

در مثال زیر ، فانکشن arrow() ، از this متد بیرونی user.sayHi() استفاده می‌کند :

```
let user = {
  firstName: "Ilya",
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi(); // Ilya
```


این ویژگی خاصی از arrow functionهاست و در مواقعی کاربرد دارد که this جداگانه‌ای نخواهیم ولی بخواهیم از this context بیرونی استفاده کنیم . در فصل‌های بعد به صورت عمیق در مورد آن صحبت خواهیم نمود .

خلاصه

- فانکشن‌هایی که در object properties ذخیره می‌شوند ، method نام دارند .
- methodها اجازه می‌دهند که objectها رفتار داشته باشند مانند : object.doSomething()
- methodها می‌توانند بوسیله this به object دسترسی داشته باشند .
- مقدار this در زمان اجرا تعیین می‌شود .
- هنگامی که یک فانکشن تعریف می‌شود ممکن است از this استفاده کند ولی مقدار واقعی this در زمان اجرا تعیین می‌شود .
- یک فانکشن می‌تواند بین چندین object کپی شود .
- وقتی methodای از یک فانکشن به این شکل فراخوانی شود : object.method() ، مقدار this در حین فراخوانی برابر است با object .
- arrow functionها خاص هستند . آنها this ندارند . اگر از this درون آنها استفاده شود ، از this کانتکست بیرونی اسفاده می‌کند .