

Getters & Setters

getters & setters

Notera: Alla objekt i JS kan ha getters och setters. Detta gäller inte bara instanser av klasser.

getters

En getter är ungefär som en funktion som inte tar emot någon parameter och som anropas utan parenteser. Det ser exakt ut som en egenskap (property) när man kommer åt den.

```
// Vanligt objekt  
let obj = {  
  get x () { return 45; }  
}
```

```
console.log(obj.x); // 45
```

```
// Klass  
class Dog {  
  get x () { return 42; }  
}
```

```
let dog1 = new Dog();  
console.log(dog1.x); // 42
```

getters

Såå... vad är poängen?

Varför inte bara använda en egenskap? Vad är skillnaden mellan:

```
class Dog {  
  get x () { return 42; }  
}
```

```
let dog1 = new Dog();  
console.log(dog1.x); // 42
```

Och:

```
class Dog {  
  constructor (data) {  
    this.x = data;  
  }  
}
```

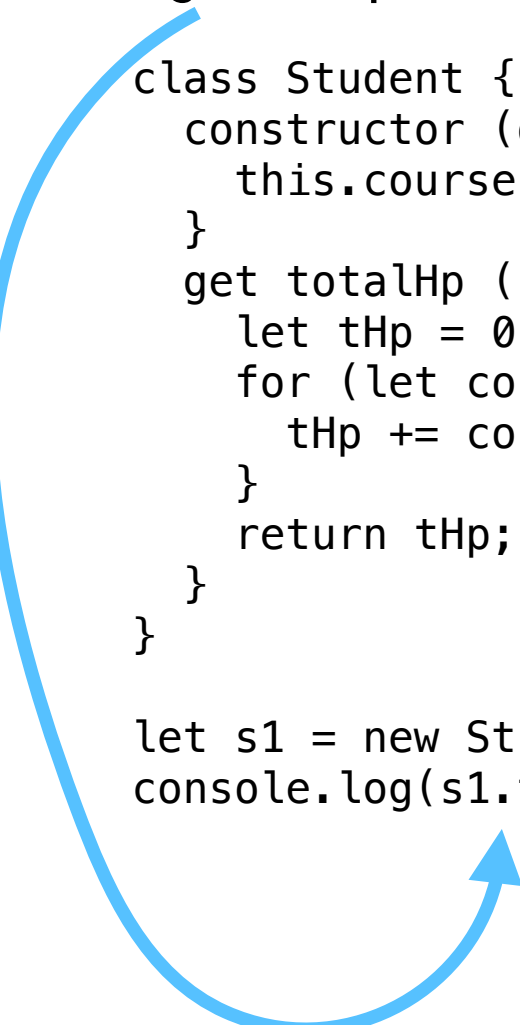
```
let dog1 = new Dog(42);  
console.log(dog1.x); // 42
```

getters

Poängen är att du kan inkludera beräkningar som genomförs varje gång vi "når" "egenskapen" (det är inte en egenskap, det är en getter, men den ser ut som en egenskap).

```
class Student {  
  constructor (data) {  
    this.courses = data;  
  }  
  get totalHp () {  
    let tHp = 0;  
    for (let course of this.courses) {  
      tHp += course.studentsHp;  
    }  
    return tHp;  
  }  
}
```

```
let s1 = new Student(courses);  
console.log(s1.totalHp); // 23.5
```



```
const courses = [  
  { code: "ME100B", hp: 15, studentsHp: 7.5 },  
  { code: "ME101B", hp: 7.5, studentsHp: 5 },  
  { code: "ME102B", hp: 7.5, studentsHp: 7.5 },  
  { code: "ME103B", hp: 7.5, studentsHp: 3.5 },  
];
```

getters

Beräkningarna kommer dessutom att baseras på de aktuella värden av egenskaperna.

```
class Student {
  constructor (data) {
    this.courses = data;
  }
  get totalHp () {
    let tHp = 0;
    for (let course of this.courses) {
      tHp += course.hpDone;
    }
    return tHp;
  }
  updateCourseHp (code, newHp) {
    // uppdaterar relevant object i this.courses.
  }
}
```

```
let s1 = new Student(courses);
console.log(s1.totalHp); // 23.5
```

```
s1.updateCourseHp("ME101B", 7.5); // Studenten har nu klarat av alla hp i ME101B
console.log(s1.totalHp); // 26 – totalHp beräknas på nytt varje gång vi efterfrågar den.
```

```
const courses = [
  { code: "ME100B", hp: 15, studentsHp: 7.5 },
  { code: "ME101B", hp: 7.5, studentsHp: 5 },
  { code: "ME102B", hp: 7.5, studentsHp: 7.5 },
  { code: "ME103B", hp: 7.5, studentsHp: 3.5 },
];
```

setters

En setter är en ganska häftig konstruktion. Den deklareraras ungefär som en metod (som tar emot en parameter). Men den "anropas" med tilldelnings-operatorn (assignment operator); dvs. =

```
class Dog {  
  set x (value) {  
    this._x = value;  
  }  
}
```

x deklareraras som en metod, fast med det reserverade ordet set.
En setter måste alltid ha exakt en parameter.

```
let dog1 = new Dog();  
dog1.x = 42;
```

En setter "anropas" såhär

setters

Varför _ (underscore / understreck)?

Notera att det blir problematiskt att skriva utan underscore.

```
class Dog {  
  set x (value) {  
    this.x = value;  
  }  
}
```

Denna tilldelning kommer att "anropa" settern igen, som kommer att köra tilldelningen, som kommer att "anropa" settern, som kommer att köra tilldelningen...

Oändlig "rekursion" (ungefär som en oändlig loop)

setters

Såå... vad är poängen?

Varför inte bara använda en egenskap? Vad är skillnaden mellan:

```
class Dog {  
  set x (value) { this._x = value; }  
}
```

```
let dog1 = new Dog();  
dog1.x = 42;  
console.log(dog1._x); // 42
```

Och:

```
class Dog {}
```

```
let dog1 = new Dog();  
dog1.x = 42;  
console.log(dog1.x); // 42
```

setters

Vertrauen ist gut, Kontrolle ist besser!

Den stora skillnaden är att du som programmerare kan exakt kontrollera vad som händer när värdet av en egenskap ändras (det är mycket vanligt att man vill validera värdet innan man tillsätter det).

```
class Dog {  
  set weight (value) {  
    if (value < 0) {  
      console.error("Yeah, I don't think so");  
      return;  
    }  
    this._weight = value;  
  }  
}  
  
let dog1 = new Dog();  
dog1.weight = -1; // Error på konsolen: "Yeah, I don't think so"  
console.log(dog1._weight); // undefined  
  
dog1.weight = 12;  
console.log(dog1._weight); // 12
```

Kombinationen getter + setter

Mycket mycket vanlig powercombo!

```
class Dog {  
  set weight (value) {  
    if (value < 0) {  
      console.error("Yeah, I don't think so");  
      return;  
    }  
    this._weight = value;  
  }  
  get weight () { return this._weight; }  
}
```

```
let dog1 = new Dog();  
dog1.weight = 12;  
console.log(dog1.weight); // 12
```

Egenskapen med understreck (weight) behålls internt inom setter:en och getter:en.

Utanför klassen använder vi `weight`, utan understreck.

Proffstips: Egenskaper som INTE ska nås från utanför klassen ska börja med understreck ()

Viktigt att ha förstått

En getter-egenskap utan setter kan inte uppdateras manuellt

```
1. class Dog {  
2.   get weight () { return this.weight; }  
3. }  
4.  
5. let dog1 = new Dog();  
6. dog1.weight = 12;  
7. console.log(dog1.weight); // undefined!
```

Problemet är oändlig rekursion på rad 2

När vi "hämtar" värdet av `dog.weight` (rad 7) så körs gettern, som i sin tur hämtar värdet av `dog.weight`, som kör gettern, som i sin tur hämtar värdet av `dog.weight`, som kör gettern...

```
1. class Dog {  
2.   get weight () { return this._weight; }  
3. }  
4.  
5. let dog1 = new Dog();  
6. dog1.weight = 12;  
7. console.log(dog1.weight); // undefined!
```

rad 6 skapar en egenskap *weight* i instansen `dog1` och tilldelar den värdet 12.

rad 7 "anropar" getter *weight*, som returnerar värdet av egenskapen *_weight*, som inte har tilldelats något värde.

```
1. class Dog {  
2.   get weight () { return this._weight; }  
3. }  
4.  
5. let dog1 = new Dog();  
6. dog1._weight = 12;  
7. console.log(dog1.weight); // 12
```

Detta skulle "fungera" eftersom vi uppdaterar egenskapen *_weight*. Men **DON'T EVER DO THIS**, för att:

- 1) egenskaper med understreck ska hållas internt till klassen.
- 2) hur ska dina kollegor veta att det är *_weight* som ska uppdateras för att *weight* ska få ett nytt värde?

Använd powercombon setter + getter istället!

Fördelar med getters & setters

Inkapsling: Getters och setters hjälper oss kontrollera åtkomsten till ett objekts egenskaper.

Validering: Setters kan inkludera valideringslogik för att säkerställa att egenskapsvärdena är korrekta.

Beräknade egenskaper: Getters kan beräkna värden dynamiskt baserat på andra egenskaper.

