Inkapsling

Fält, ”getters”, ”setters” och privata medlemmar

Tärningen är kastad – det andra försöket

- ✓ Det finns problem…

```
// die.js
export class Die {
  constructor (maxFaceValue = 6) {
    this.maxFaceValue = maxFaceValue
    this.roll()
  }

roll () {
    this.faceValue = Math.floor(Math.random() * this.maxFaceValue) + 1
  }
}
```

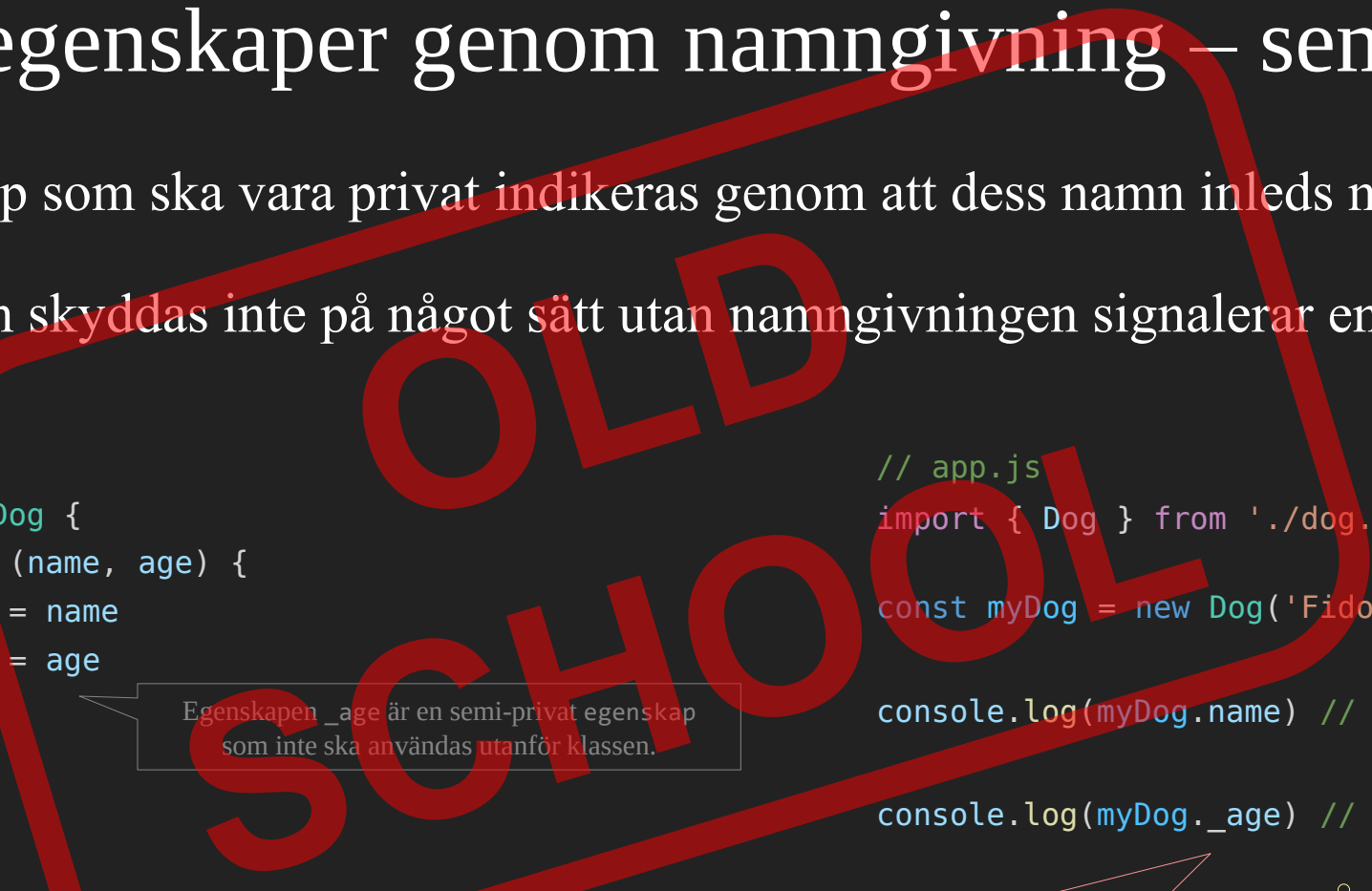
Inte bra att det hur som helst går att ändra på egenskapers värden!

- ✓ …med att egenskapernas värden kan sättas till vilket värde som helst!
  - Går det att dölja, skydda, en del av ett objekts data för omvärlden? JA!

```
// app.js
import { Die } from './die.js'

const die = new Die()
console.log(die.faceValue)    // OUTPUT: 4(?)
console.log(die.maxFaceValue) // OUTPUT: 6

die.faceValue = 6
console.log(die.faceValue)    // OUTPUT: 6

die.faceValue = 'This is a strange value for a die!'
console.log(die.faceValue)    // OUTPUT: This is a strange value for a die!

die.maxFaceValue = 'twentyish'
die.roll()
console.log(die.faceValue)    // NaN
```

Privata egenskaper genom namngivning – semi-privat

- ✓ En egenskap som ska vara privat indikeras genom att dess namn inleds med ett understreck ’_’.
- ✓ Egenskapen skyddas inte på något sätt utan namngivningen signalerar endast ”Använd inte denna egenskap!”




```
// dog.js
export class Dog {
  constructor (name, age) {
    this.name = name
    this._age = age
  }
}
```

Egenskapen `_age` är en semi-privat egenskap som inte ska användas utanför klassen.

```
// app.js
import { Dog } from './dog.js'

const myDog = new Dog('Fido', 3)

console.log(myDog.name) // OUTPUT: Fido

console.log(myDog._age) // BAD PROGRAMMER, NO DONUT!
```

Publika egenskaper går självklart bra att använda.

Använd ALDRIG en semi-privat egenskap utanför en klass!

Ett annat sätt att åstadkomma en privat egenskap är att använda `WeakMaps` med vars hjälp data kan skyddas på ett helt annat sätt, MEN det är mycket mer komplicerat…

Hur ska hundens ålder kommas åt om egenskapen `_age` inte får användas utanför klassen?

Privata fält

- ✓ Ett fält som ska vara privat skapas genom att dess namn inleds med ett nummertecken (#).
- ✓ Fältet skyddas och kan endast användas i klassdefinitionen.

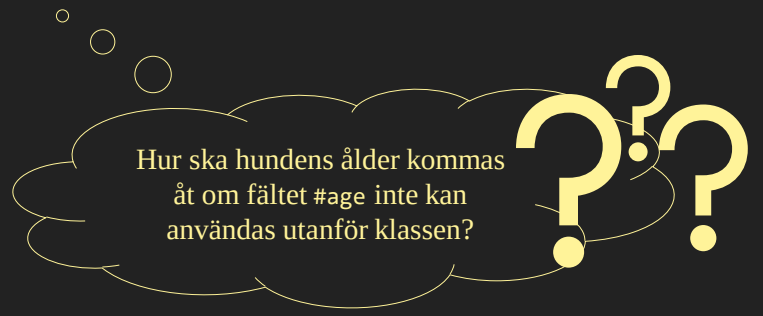
```
// dog.js
export class Dog {
    #age // Private field!

    constructor (name, age) {
        this.name = name
        this.#age = age
    }
}
```

Fältet `#age` är privat och kan inte användas utanför klassen. Inuti klassen går det däremot bra!

```
// app.js
import { Dog } from './dog.js'

const myDog = new Dog('Fido', 3)

console.log(myDog.name) // OUTPUT: Fido

console.log(myDog.#age) // BAD PROGRAMMER, NO DONUT!
```

Publika egenskaper går självklart bra att använda.

Du kan ALDRIG använda ett privat fält utanför en klass!



Accessor- och mutator-metoder - ”getters” och ”setters”

- ✓ En accessor-metod binder en egenskap till en funktion som anropas då en egenskaps värde hämtas.
  - En ”getter” skapas genom att före definitionen av en metod använda modifieraren `get`.
- ✓ En mutator-metod binder en egenskap till en funktion som anropas då en egenskaps värde sätts.
  - En ”setter” skapas genom att före definitionen av en metod använda modifieraren `set`.

```
export class Dog {
  #age // private field

  constructor (name, age) {
    this.name = name
    this.age = age
  }

  get age () {
    return this.#age
  }

  set age (value) {
    if (!Number.isInteger(value) || value < 1) {
      throw new Error('Invalid age.')
    }
    this.#age = value
  }
}
```

Leder till att ”setter”-funktionen för egenskapen `age` anropas.

Denna ”getter” anropas då egenskapen `age` värde hämtas; `console.log(`Min hund är ${myDog.age} år.`).`

Denna ”setter” anropas då egenskapen `age` tilldelas ett värde; `myDog.age = 3.`

Skydda data med ”setter”

- ✓ För att skydda ett privats fälts värde kan den kapslas med ”getter” och ”setter”.

```
// dog.js
export class Dog {
  #age // private field

  constructor (name, age) {
    this.name = name
    this.age = age
  }

  get age () {
    return this.#age
  }

  set age (value) {
    if (!Number.isInteger(value) || value < 1) {
      throw new Error('Invalid age.')
    }
    this.#age = value
  }
}
```

”getter”

”setter”

- ✓ Klarar inte ett värde valideringen ”setter”-funktionen utför kastas ett undantag.

”getter”-funktionen för age anropas.

”setter”-funktionen för age anropas.

Ogiltiga värden för age!

```
// app.js
import { Dog } from './dog.js'

const myDog = new Dog('Fido', 3)
console.log(myDog.name) // OUTPUT: Fido
console.log(myDog.age)  // OUTPUT: 3

myDog.age =  7
console.log(myDog.age) // OUTPUT: 7

myDog.age = 3.14     // ERROR: Error: Invalid age.
myDog.age = 0        // ERROR: Error: Invalid age.
myDog.age = 'VET EJ' // ERROR: Error: Invalid age.
```

Tärningen är kastad – det tredje försöket

- ✓ Fälten `#faceValue` och `#maxFaceValue` är privata och kapslas in och görs tillgängliga genom ”getters” och ”setters”.

```
// die.js
export class Die {
  #faceValue
  #maxFaceValue

  constructor (maxFaceValue = 6) {
    this.maxFaceValue = maxFaceValue
    this.roll()
  }

  static from (other) {
    return new Die(other._maxFaceValue)
  }

  get faceValue () {
    return this.#faceValue
  }

  get maxFaceValue () {
    return this.#maxFaceValue
  }

  set maxFaceValue (value) {
    if (!Number.isInteger(value) || value < 1) {
      throw new Error('Invalid number of faces.')
    }

    this.#maxFaceValue = value
  }

  roll () {
    this.#faceValue = Math.floor(Math.random() * this.#maxFaceValue) + 1
  }
}
```

Det går inte att tilldela `faceValue` ett nytt värde, bara hämta det.

Det går att tilldela `maxFaceValue` ett nytt värde om det klarar valideringen, men inte annars. (Det går även att hämta värdet.)

- ✓ Undantag kastas om egenskaper används på fel sätt.

```
// app.js
import { Die } from './die.js'

const die = new Die()
console.log(die.faceValue)    // OUTPUT: 4(?)
console.log(die.maxFaceValue) // OUTPUT: 6

// ERROR: TypeError: Cannot set property faceValue of #<Die> which has only a getter
die.faceValue = 6
console.log(die.faceValue)    // OUTPUT: 6

// ERROR: TypeError: Cannot set property faceValue of #<Die> which has only a getter
die.faceValue = 'This is a strange value for a die!'
console.log(die.faceValue)    // OUTPUT: This is a strange value for a die!

// Error: Invalid maximum number of faces.
die.maxFaceValue = 'twentyish'
die.roll()
console.log(die.faceValue)    // NaN
```