JavaScript - OOP

Ingo Köster

Diplom-Informatiker (FH)

Klassen & Konstruktoren

› Mit dem Schlüsselwort `class` wird eine Klasse definiert

› Mit dem Schlüsselwort `constructor` wird innerhalb dieser Klasse eine Konstruktorfunktion erstellt
  › Es darf nur einen Konstruktor geben!

```
class Kreis {
  constructor(r) { this.radius = r; }
}
let k1 = new Kreis(1);
```

Eigenschaften der Klasse

```
class Animal {
  constructor(name = 'anonymous', legs = 4, noise =
 'nothing') {
    this.type = 'animal';
    this.name = name;
    this.legs = legs;
    this.noise = noise;
  }
}
```

Alternativen um Eigenschaften anzulegen

```
class MyClass
{
  eins = 1;
  zwei = 2;
  drei = 3;
}
```

```
class MyClass
{
  constructor()
  {
    this.eins = 1;
    this.zwei = 2;
    this.drei = 3;
  }
}
```

Funktionen

```
class Kreis {
    constructor(r)
    {
        this.radius = r;
    }
    berechneFläche()
    {
        return Math.PI * Math.pow(this.radius, 2);
    }
}
```

Hinweise

- Klassendeklarationen werden immer im Strict Mode ausgeführt
- Es ist daher nicht erforderlich in Klassen `"use strict"` hinzuzufügen
- Hoisting: Im Gegensatz zu Funktionen werden Klassendeklarationen nicht „hochgezogen“
- Bedeutet, dass eine Klasse deklariert werden muss, bevor sie verwendet werden kann

```
new MyClass();

class MyClass {}
▶Uncaught ReferenceError: MyClass is not defined
```

Getter und Setter ähnlich zu C#

```
class Person {
  constructor(name) {
    this.name = name;
  }
  get pname() {
    return this.name;
  }
  set pname(x) {
    this.name = x;
  }
}
```

› `let p = new Person("Tim");`

› Lesen

  › `let name = p.pname;`

› Schreiben

  › `p.pname = "Tom";`

Vererbung

› Mittels `extends` kann von einer anderen Klasse geerbt werden
  › in C# der Doppelpunkt

```
class Fahrzeug { … }                    // Elternklasse
class PKW extends Fahrzeug { … }        // Kindklasse
class LKW extends Fahrzeug { … }        // Kindklasse
```

› Von den Standardobjekten in JavaScript wie `Array`, `Date`, etc. kann ebenfalls geerbt werden

Vererbung – Konstruktorverkettung

› Die Konstruktorverkettung wird mittels des Schlüsselworts `super()` erreicht (in C# `base`)

```
class Fahrzeug
{
  constructor(kmStand) { … }
}

class PKW extends Fahrzeug {
  constructor(kmStand)
  {
     super(kmStand);        // Ruft constructor in Fahrzeug auf
  }
}
```

super ähnlich zu base

```
class A
{
  hallo() {
    return "Hallo A";
  }
  ausgabe() {
   console.log(this.hallo());
  }
}
```

```
class B extends A
{
 hallo()
 {
   return
   super.hallo() + " & B";
 }
}
```

Statische Eigenschaften

```
class MyClass
{
  x = 1;
  y = 2;
  static z = 3;
}
console.log(MyClass.z);
```

Statische Methoden

```
class Punkt
{
  constructor(x, y) { … }
  static abstand(a, b) { … return Math.sqrt(…); }
}

let p1 = new Punkt(5, 5);
let p2 = new Punkt(10, 10);
console.log(Punkt.abstand(p1, p2));
```

Private Eigenschaften

› Private Eigenschaften mittels der Raute anlegen und verwenden

```
class MyClass
{
 a = 1;                 // public
 #b = 2;                // private
 static #c = 3;         // private
                        und static
 erhöheB() { this.#b++; }
}
```

› Verwendung

```
let m = new MyClass();
m.erhöheB();    // OK
m.#b = 0;       // Fehler
```

Private Funktionen

› Private Funktionen werden ebenfalls mit der Raute angelegt

```
class MyClass {
  #privateMethode() {
    return 'Hallo Welt';
  }

  publicMethode() {
    return this.#privateMethode();
  }
}
```

› Private statische Funktionen sind ebenfalls möglich
  › Konstruktoren können nicht privat sein

Private Getter und Setter

› Auch private Getter und Setter verwenden die Raute

```
class Person
{
  get #name()
  {
    return "Tim";
  }
}
```

Vererbung

- Private Eigenschaften oder Funktionen werden vererbt
- Auf private Eigenschaften oder Funktionen kann in einer Vererbungshierarchie nicht zugegriffen werden
- Protected wie in C# existiert in JavaScript nicht