1 Errata

1.1 Seite 43: Reguläre Ausdrücke

Statt Backslash müsste es Slash heißen: "... und enden jeweils mit einem Slash (/).“.

(Behoben im 1. korrigierten Nachdruck 2015)

1.2 Seite 71: Lokale vs. globale Variable

In Listing 2.15 sollte die Variable `name` ohne "var" angelegt werden. Unter Node.js beispielsweise gibt die Funktion `getNameGlobal()` ansonsten `undefined` zurück (vgl. Seite 45).

```
name = "globaler Name";
function getNameGlobal() {
  return this.name;
}
console.log(getNameGlobal());
```

(Behoben im 1. korrigierten Nachdruck 2015)

1.3 Seite 156: Nichtprivilegierte öffentliche Methoden

Die Implementierung in Listing 3.34 ist (in diesem Kontext zumindest) nicht ganz richtig. Prinzipiell kann man natürlich über Closures private Variablen emulieren, allerdings führt das im jetzigen Beispiel dazu, dass man nicht mehr wirklich einzelne (verschiedene) Objektinstanzen erzeugen kann bzw. dass, wenn man es doch macht, die "Instanzvariablen" keine solchen sind und sich Änderungen auf jeweils andere Objektinstanzen auswirken:

```
var max = new Mitarbeiter('Max', 'Mustermann', 2345);
var moritz = new Mitarbeiter('Moritz', 'Mustermann', 2346);
console.log(max.getName()); // Moritz
console.log(moritz.getName()); // Moritz
```

Listing 1.1: Das Problem der beschriebenen Implementierung: die einzelnen Objektinstanzen teilen sich die „Instanzvariablen“ (die aus diesem Grund keine solchen sind)

Nichtprivilegierte öffentliche Methoden in Kombination mit Closures geben aber trotzdem Sinn. Um die Vorteile davon deutlich zu machen, müssen aber auch die vorausgehenden Beispiele im Buch etwas angepasst werden. Und zwar wie folgt:

Die Implementierung in Listing 3.31 auf Seite 154 („Privilegierte öffentliche Methoden“) wird um die Methode `print()` ergänzt:

```
function Mitarbeiter(name, nachname, mitarbeiterID) {
    var name = name;
    var nachname = nachname;
    var mitarbeiterID = mitarbeiterID;
    this.getName = function() {
        return name;
    }
    this.getNachname = function() {
        return nachname;
    }
    this.getMitarbeiterID = function() {
        return mitarbeiterID;
    }
```

```
    this.setName = function(neuerName) {
        name = neuerName;
    }
    this.setNachname = function(neuerNachname) {
        nachname = neuerNachname;
    }
    this.setMitarbeiterID = function(neueMitarbeiterID) {
        mitarbeiterID = neueMitarbeiterID;
    }
    this.print = function() {
      return this.getName() + ' '
            + this.getNachname()
            + ' (' + this.getMitarbeiterID() + ')';
    };
}
```

Listing 1.2: Privilegierte öffentliche Methoden

Diese Implementierung hat den Vorteil, dass Instanzvariablen privat sind, allerdings den Nachteil, dass die Methode `print()` für jede Objektinstanz definiert wird (siehe Abbildung 1.1).

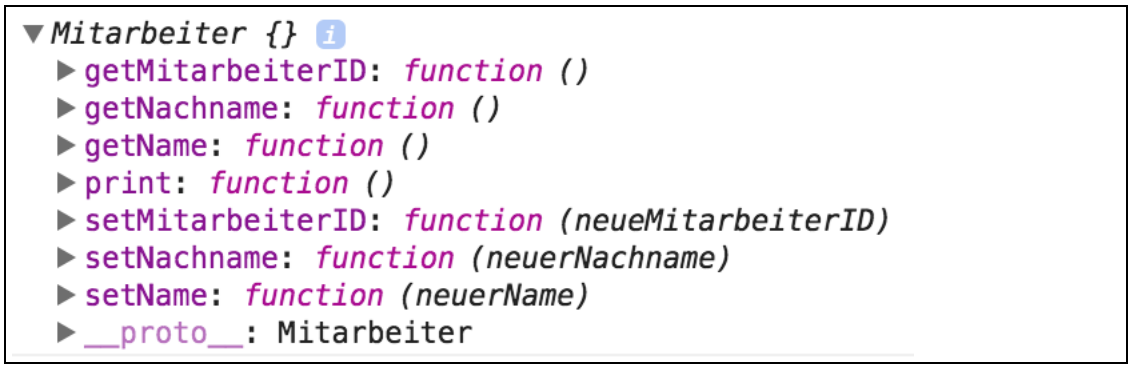

Abbildung 1.1: Bei der Verwendung privilegierter öffentlicher Methoden sind die Instanzvariablen zwar privat, die Methode `print()` jedoch pro Objektinstanz vorhanden

Die zweite Anpassung betrifft die Implementierung in Listing 3.32 auf Seite 155 („Nichtprivilegierte öffentliche Methoden“), die ebenfalls um die Methode `print()` ergänzt wird, und zwar wie folgt:

```
function Mitarbeiter(name, nachname, mitarbeiterID) {
    this._name = name;
    this._nachname = nachname;
    this._mitarbeiterID = mitarbeiterID;
}
Mitarbeiter.prototype.getName = function() {
    return this._name;
};
Mitarbeiter.prototype.getNachname = function() {
    return this._nachname;
};
Mitarbeiter.prototype.getMitarbeiterID = function() {
    return this._mitarbeiterID;
```

```
};
Mitarbeiter.prototype.print = function() {
  return this.getName() + ' ' + this.getNachname() + ' (' + this.getMitarbeiterID() +
')';
};
```

Listing 1.3: Nichtprivilegierte öffentliche Methoden

Das wiederum hat den Vorteil, dass die Methode `print()` nur einmal am Protoypen definiert wird, allerdings zugleich den Nachteil, dass die Instanzvariablen nicht mehr privat sind (siehe Abbildung 1.2).

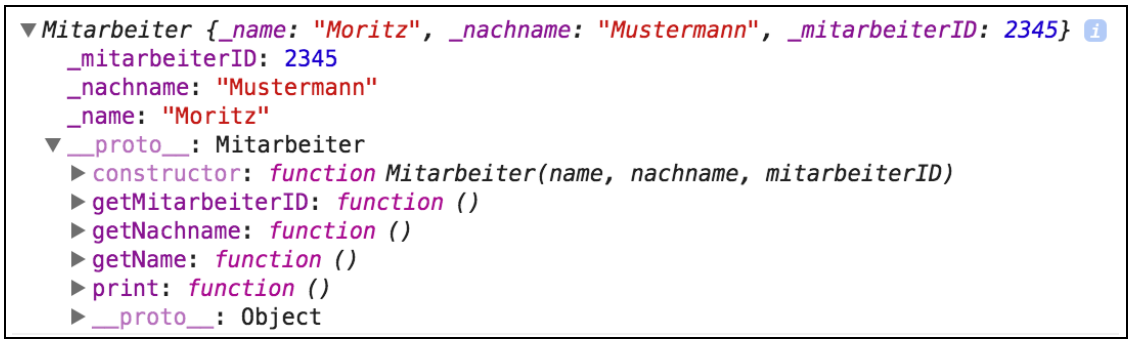

Abbildung 1.2: Bei der Verwendung nichtprivilegierter öffentlicher Methoden ist die Methode `print()` nicht an jeder Objektinstanz vorhanden, jedoch sind die Instanzvariablen nicht privat

Schlussendlich – und jetzt sollte gleich der Vorteil von nichtprivilegierten öffentlichen Methoden in Kombination mit Closures deutlich werden – wird die Implementierung aus Listing 3.34 auf Seite 156 („Private Eigenschaften und nichtprivilegierte öffentliche Methoden über Closure und IIFE“) wie folgt geändert:

```
var Mitarbeiter = (function() {
    function Mitarbeiter(name, nachname, mitarbeiterID) {
      var name = name;
      var nachname = nachname;
      var mitarbeiterID = mitarbeiterID;
      this.getName = function () {
        return name;
      };
      this.getNachname = function () {
        return nachname;
      };
      this.getMitarbeiterID = function () {
        return mitarbeiterID;
      };
    }
    return Mitarbeiter;
})();
Mitarbeiter.prototype.print = function() {
  return this.getName() + ' ' + this.getNachname() + ' (' + this.getMitarbeiterID() + ')';
};
```

Listing 1.4: Private Eigenschaften und nichtprivilegierte öffentliche Methoden über Closure und IIFE

Diese Implementierung kombiniert die Vorteile der beiden vorausgehenden Implementierungen: die Methode `print()` wird nur einmal am Prototypen definiert, die Instanzvariablen sind aber privat (siehe Abbildung 1.3). Der Zugriff von der nichtprivilegierten Methode erfolgt dabei über die privilegierten Methoden (die allerdings immer pro Objektinstanz vorhanden sein müssen).

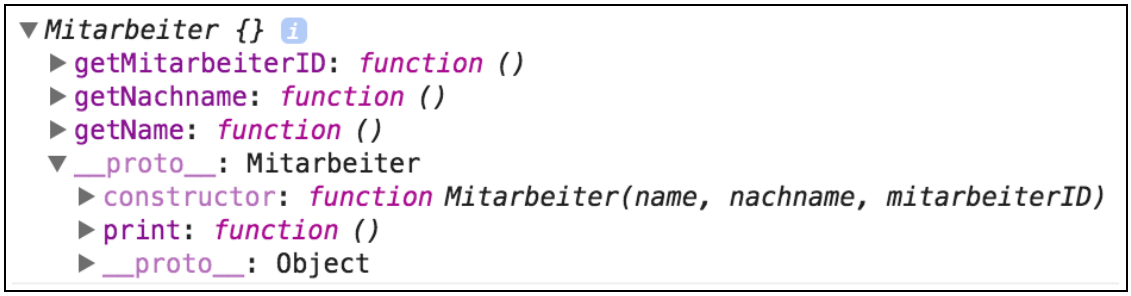

Abbildung 1.3: Bei der Verwendung nichtprivilegierter öffentlicher Methoden in Kombination mit einer Closure ist die Methode `print()` nicht an jeder Objektinstanz vorhanden, die Instanzvariablen aber trotzdem privat

(Behoben im 1. korrigierten Nachdruck 2015)

1.4 Seite 181, „defineProperty“

`defineProperty()` heißt immer genau so, auch wenn der Fehlerteufel uns einmal „definePropery()“ untergejubelt hat.

(Behoben im 1. korrigierten Nachdruck 2015)

1.5 Seite 212 und folgende, Konstruktorfunktion für Proxies

Es ist mittlerweile nicht mehr möglich, bei der Erzeugung von Proxies das Schlüsselwort `new` wegzulassen. Statt `Proxy()` muss es daher in Listing 4.70 und in den folgenden Listings immer `new Proxy()` (mit entsprechenden Argumenten) heißen.

1.6 Seite 216, Wert statt Eigenschaft

Anstatt die Eigenschaft bzw. den Eigenschaftsnamen über `isInteger()` zu überprüfen, muss in Listing 4.74 natürlich der konkrete Wert der Eigenschaft überprüft werden:

```
let personValidator = {
  set: function(objekt, eigenschaft, wert) {
    if (eigenschaft === 'alter') {
      if (!Number.isInteger(wert)) {
        throw new TypeError('Das Alter muss eine Zahl sein.');
      }
    }
    objekt[eigenschaft] = wert;
  }
};
```

Listing 1.5: Überprüfen des Wertes