2.3 - JSA: Vererbung in JavaScript

Einleitung

In der objektorientierten Programmierung ist Vererbung ein zentrales Konzept: Du kannst vorhandene Klassen erweitern, spezialisieren oder Methoden überschreiben. JavaScript unterstützt seit ES6 eine vereinfachte, klassenbasierte Vererbung über die Schlüsselwörter `extends` und `super()`.

1. Klassenvererbung mit `extends`

Um eine Klasse zu erweitern, verwendest du `extends`.

```
class Vehicle {
  constructor(id) {
    this.id = id;
  }

  getId() {
    return this.id;
  }
}

class Bus extends Vehicle {
  constructor(id, seats) {
    super(id); // ruft den Konstruktor von Vehicle auf
    this.seats = seats;
  }
}

const b = new Bus("B123", 40);
console.log(b.getId()); // "B123"
console.log(b.seats);   // 40
```

Wichtig:

- `super(...)` **muss** in einem Konstruktor aufgerufen werden, bevor `this` verwendet werden darf.
- Die Basisklasse (Vehicle) bleibt unverändert – sie wird **nicht überschrieben**.

Python-Vergleich

```
class Vehicle:
    def __init__(self, id):
        self.id = id

class Bus(Vehicle):
    def __init__(self, id, seats):
        super().__init__(id)
        self.seats = seats
```

Beide Sprachen verwenden `super()`, allerdings ist der Aufruf in JavaScript **obligatorisch**, wenn du einen eigenen Konstruktor in der Kindklasse definierst.

2. Methodenüberschreibung und Shadowing

Wenn du in einer abgeleiteten Klasse eine Methode mit gleichem Namen wie in der Basisklasse definierst, wird die neue Methode verwendet. Man spricht von **Shadowing**:

```
class Vehicle {
  greet() {
    console.log("Hello from Vehicle");
  }
}

class Car extends Vehicle {
  greet() {
    console.log("Hello from Car");
  }
}

const c = new Car();
c.greet(); // "Hello from Car"
```

Du kannst aber innerhalb der Kindklasse auch explizit die Methode der Elternklasse aufrufen:

```
class Car extends Vehicle {
  greet() {
    super.greet(); // greift auf Elternmethode zu
    console.log("... and hello from Car");
  }
}
```

Praxis-Tipp:

- `super.methodName()` funktioniert **nur innerhalb von Methoden** der abgeleiteten Klasse.

Python-Vergleich

```
class Car(Vehicle):
    def greet(self):
        super().greet()
        print("... and hello from Car")
```

3. Vererbung von Konstruktorfunktionen

Auch Konstruktorfunktionen (aus „Pre-ES6-Zeiten“) können als Basisklasse verwendet werden:

```
function Device(name) {
  this.name = name;
  this.describe = function() {
    console.log("Device: " + this.name);
  };
}

class Phone extends Device {
  constructor(name, os) {
    super(name);
    this.os = os;
  }
}

const p = new Phone("Galaxy", "Android");
p.describe(); // "Device: Galaxy"
```

Das ist möglich, weil `class` intern ohnehin syntaktischer Zucker über die Prototypenkette ist.

4. `instanceof` und `constructor.name`

Wie kannst du prüfen, ob ein Objekt von einer bestimmten Klasse abstammt?

Mit `instanceof`

```
const car = new Car();
console.log(car instanceof Car);      // true
console.log(car instanceof Vehicle);  // true
```

`instanceof` geht die gesamte Vererbungskette entlang. Ist irgendwo in der Kette die angegebene Klasse, wird `true` zurückgegeben.

Mit `constructor.name`

```
console.log(car.constructor.name); // "Car"
```

Diese Variante gibt dir den Namen der Klasse zurück, funktioniert aber **nicht bei anonymen Klassen**.

5. Zusammenfassung & Best Practices

- Verwende `extends` für klassische Vererbung von Klassen
- Nutze `super(...)` im Konstruktor der Kindklasse
- Methoden können überschrieben werden (Shadowing)
- `super.method()` ruft Elternmethode auf
- `instanceof` ist ideal für Typprüfungen zur Laufzeit
- Vermeide zu tiefe Vererbungshierarchien – **Komposition > Vererbung**, wenn sinnvoll

6. Übungsaufgaben

Aufgabe 1: Vererbung nachbauen

Erstelle eine Klasse `Animal` mit einer Methode `speak()`. Erstelle dann eine Klasse `Dog`, die diese Methode überschreibt.

Aufgabe 2: Fehleranalyse

Was passiert bei folgendem Code?

```
class Child extends Parent {
  constructor() {
    this.age = 5;
    super();
  }
}
```

Aufgabe 3: Konstruktorfunktionen erben

Baue eine Klasse `Printer`, die von einer Funktion `Device(name)` erbt. Implementiere eine Methode `print()`.

Aufgabe 4: instanceof-Test

Erstelle eine kleine Vererbungskette und prüfe mit `instanceof`, ob die Instanz den Erwartungen entspricht.

Aufgabe 5: Methodenschatten

Erstelle eine Klasse `Person` mit einer Methode `greet()`, und eine Klasse `Employee`, die `greet()` überschreibt, aber auch `super.greet()` verwendet.

7. Micro-Projekt: Fahrzeughierarchie mit Spezialisierungen

Ziel

Modelliere eine Fahrzeughierarchie mit Basis- und Spezialisierungsklassen.

Anforderungen

- `Vehicle`: id, position
- `Bus`: Eigenschaft `seats`
- `Taxi`: Eigenschaft `licenseNumber`, Methode `calculateFare()`
- Überschreibe ggf. `getInfo()` in jeder Klasse
- Verwende `super()` zur Initialisierung

Beispiel

```
class Vehicle {
  constructor(id, position) {
    this.id = id;
    this.position = position;
  }

  getInfo() {
    return `ID: ${this.id}`;
  }
}

class Taxi extends Vehicle {
  constructor(id, position, license) {
    super(id, position);
    this.license = license;
  }

  getInfo() {
    return `${super.getInfo()}, License: ${this.license}`;
  }
}
```

Erweiterungsidee

- Erstelle eine Funktion `describeFleet(arrayOfVehicles)`, die mit `instanceof` prüft, welche Klasse vorliegt, und entsprechend `getInfo()` aufruft.