2.1 - JSA: Klassen in JavaScript – Grundlagen & Konstruktoren

Einleitung

Mit der Einführung von **Klassen** in ECMAScript 6 (ES6) wurde JavaScript um eine vertraute Struktur aus vielen anderen Programmiersprachen erweitert.

Die Nutzung von Klassen verbessert die Lesbarkeit und Struktur komplexer Anwendungen und erlaubt es, objektorientierte Konzepte wie Kapselung, Vererbung und Polymorphie in JavaScript klarer umzusetzen.

1. Was ist eine Klasse in JavaScript?

Eine Klasse ist eine **Vorlage (Template)** für die Erstellung von Objekten mit gemeinsamen Eigenschaften und Methoden. Du kannst dir eine Klasse wie eine Schablone vorstellen, anhand derer beliebig viele gleichartige Objekte erzeugt werden.

Früher wurden in JavaScript Objekte überwiegend mit Konstruktorfunktionen oder Objektliteralen erstellt. Klassen bieten nun eine elegantere und besser strukturierte Möglichkeit, das gleiche Ziel zu erreichen.

Klassenbasierte Objektinstanziierung

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

const car = new Vehicle("AB1234");
console.log(car.id); // "AB1234"
```

Vergleich zu Python

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

car = Vehicle("AB1234")
print(car.id)  # "AB1234"
```

Gemeinsamkeiten:

- Beide Sprachen verwenden `class` zur Definition.
- Konstruktoren heißen `constructor` (JS) bzw. `__init__` (Python).
- Instanzvariablen werden über `this` (JS) bzw. `self` (Python) zugewiesen.

Unterschiede:

- In JavaScript muss der Konstruktor explizit `constructor` heißen.
- `this` ist in JS kontextabhängig und kann verloren gehen (z. B. bei Callbacks), während `self` in Python als Parameter explizit ist und somit robuster.

---

2. Klassendeklaration vs. Klassenausdruck

JavaScript erlaubt es, Klassen auf zwei Arten zu definieren: per Deklaration oder per Ausdruck.

Klassendeklaration

```
class MyClass {
  constructor() {
    console.log("Instanz erzeugt");
  }
}
```

Diese Variante ist sehr übersichtlich und sollte bevorzugt verwendet werden, wenn du eine Klasse **global oder lokal verfügbar machen willst**, bevor sie im Code verwendet wird. Klassen-Deklarationen sind **hoisted** – das heißt, sie werden vom JavaScript-Interpreter vor der Ausführung an den Anfang des Scopes gezogen (ähnlich wie Funktionsdeklarationen).

Klassenausdruck

```
const MyClass = class {
  constructor() {
    console.log("Instanz erzeugt");
  }
};
```

Ein Klassenausdruck ist nützlich, wenn du eine Klasse **nur innerhalb eines bestimmten Kontextes** brauchst – etwa innerhalb einer Funktion oder um dynamisch eine Klasse zu erzeugen. Klassenausdrücke sind **nicht hoisted** und müssen daher definiert sein, bevor sie verwendet werden.

Wann welche Variante verwenden?

- **Deklaration:** Bei klassischen Klassen, die du mehrfach verwenden oder exportieren möchtest (Standardfall).
- **Ausdruck:** Für dynamische oder anonyme Klassen (z. B. in Factory-Funktionen, Closures oder Modulen).

Zusatz: Auch benannte Klassenausdrücke sind möglich – so lässt sich innerhalb der Klasse auf den Namen referenzieren, was bei Rekursion hilfreich sein kann.

---

3. Konstruktor und Methoden

Der **Konstruktor** ist eine spezielle Methode, die automatisch aufgerufen wird, wenn mit `new` ein Objekt erstellt wird. Hier legst du initiale Werte für das Objekt fest.

Struktur einer Klasse mit Methoden

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

  setStatus(newStatus) {
    this.status = newStatus;
  }

  getStatus() {
    return this.status;
  }
}

const bike = new Vehicle("X1023");
bike.setStatus("ready");
console.log(bike.getStatus()); // "ready"
```

Warum Methoden nicht im Konstruktor definieren?

Methoden im Konstruktor erzeugen **pro Instanz eine neue Funktion** im Speicher – das ist ineffizient. Stattdessen sollten Methoden **am Prototyp** hängen, also direkt im Rumpf der Klasse definiert sein. So teilen sich alle Instanzen dieselbe Methode.

Python-Vergleich

```python
class Vehicle:
    def __init__(self, id):
        self.id = id
        self.status = "offline"

    def set_status(self, new_status):
        self.status = new_status

    def get_status(self):
        return self.status
```

Unterschiede:

- Python verwendet `self` explizit in jeder Methodensignatur.
- In JavaScript ist `this` implizit, aber kann verloren gehen, z. B. bei Callbacks – deshalb sind Arrow-Funktionen manchmal hilfreicher.

---

4. Zusammenfassung & Best Practices

- Verwende **Klassen statt Konstruktorfunktionen**, wenn du mit OOP arbeitest.
- Nutze **Methodendefinitionen außerhalb des Konstruktors**, um Speicher zu sparen.
- Benenne Klassen mit **PascalCase** (z. B. `MyClass`).
- Initialisiere alle Pflichtattribute im Konstruktor.
- Verwende Arrow Functions, wenn `this` konsistent erhalten bleiben soll (z. B. bei Eventhandlern).

---

5. Übungsaufgaben

Aufgabe 1: Begriffsklärung

Erkläre in eigenen Worten:

- Was ist eine Klasse?
- Was ist ein Konstruktor?
- Was bedeutet `this` in einer Klassenmethode?

Aufgabe 2: Fehleranalyse

Was ist an folgendem Code falsch?

```javascript
class Car {
  constructor(id) {
    this.id = id;
  },
  drive() {
    console.log("Driving...");
  }
}
```

Tipp: Achte auf die Trennzeichen im Klassenrumpf.

Aufgabe 3: Eigene Klasse schreiben

Erstelle eine Klasse `Book`, die folgende Eigenschaften hat:

- Titel
- Autor
- Methode `describe()` gibt einen Text wie "Titel von Autor" zurück

Aufgabe 4: Konstruktorfunktion in Klasse umwandeln

Wandle diesen Code in eine Klasse um:

```
function Lamp(color) {
  this.color = color;
  this.turnOn = function() {
    console.log("Lamp on!");
  }
}
```

Aufgabe 5: Python zu JavaScript

Übersetze folgenden Python-Code in JavaScript:

```
class User:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello, " + self.name)
```

Zusatzaufgaben:

- Erkläre den Unterschied zwischen Klassendeklaration und Klassenausdruck.
- Was ist der Unterschied zwischen Instanz- und Klassenmethoden?

---

6. Micro-Projekt: Fahrzeug-Basisobjekt

Ziel

Entwickle eine JavaScript-Klasse `Vehicle`, die Fahrzeuge modelliert. Du sollst erste OOP-Konzepte praktisch anwenden.

Anforderungen

- Konstruktor mit `id`, `status`, `latitude`, `longitude`
- Methoden:
  - `getPosition()`: gibt Position als Objekt zurück
  - `setPosition(latitude, longitude)`: aktualisiert Position
  - `updateStatus(status)`: ändert Status ("free", "busy", ...)

Beispiel

```
const taxi = new Vehicle("TX204", "free", 59.32, 18.06);
taxi.setPosition(59.34, 18.09);
console.log(taxi.getPosition()); // { latitude: 59.34, longitude: 18.09 }
```

Optional

- Schreibe zusätzlich eine Python-Version der Klasse `Vehicle`.
- Stelle Gemeinsamkeiten und Unterschiede in einer Tabelle gegenüber.