Klassen, Interfaces und Vererbung in TypeScript

Klassen

Klassen sind ein grundlegendes Konzept der objektorientierten Programmierung. Sie definieren die Eigenschaften und Methoden, die Objekte dieses Typs besitzen.

TypeScript unterstützt Access Modifiers, um die Sichtbarkeit von Eigenschaften und Methoden zu steuern:

- `public`: Die Eigenschaft oder Methode ist überall zugänglich.
- `private`: Die Eigenschaft oder Methode ist nur innerhalb der Klasse sichtbar.
- `protected`: Die Eigenschaft oder Methode ist innerhalb der Klasse und in Unterklassen zugänglich.

Außerdem unterstützt TypeScript statische Eigenschaften und Methoden, Getters und Setters sowie abstrakte Klassen.

Vererbung

Vererbung ermöglicht das Wiederverwenden von Code in abgeleiteten Klassen. Die Unterklasse erbt alle Eigenschaften und Methoden der Basisklasse und kann diese erweitern oder überschreiben.

Interfaces

Interfaces definieren eine Struktur, die von Klassen implementiert werden kann. Sie legen fest, welche Eigenschaften und Methoden eine Klasse besitzen muss.

Typen vs. Interfaces

- `interface` wird für Strukturdefinitionen verwendet.
- `type` ist vielseitiger und eignet sich für komplexe Typkompositionen.

Interfaces können durch mehrere Deklarationen oder `extends` erweitert werden, während Types nicht erneut definiert werden können, aber mittels Intersection (`&`) kombiniert werden können.

Index Signaturen

Index Signaturen ermöglichen es, beliebige Schlüssel-Wert-Paare in Objekten zu speichern.

Type Aliase

Ein Type Alias ist ein Name für einen beliebigen Typ.

Coding Kata

1. Erstelle die Basisklasse `Person` mit `name` (string) und `age` (number) und einer Methode `getInfo()`, die `"Name: [name], Alter: [age]"` zurückgibt.
2. Erstelle die abgeleitete Klasse `Employee`, die von `Person` erbt und eine zusätzliche Eigenschaft `position` (string) hat. Überschreibe `getInfo()`, um `position` hinzuzufügen.
3. Erstelle das Interface `IContact` mit `email` (string) und einer Methode `getContactInfo()`, die die Kontaktinformationen als Zeichenkette zurückgibt.
4. Implementiere `IContact` in `Employee`.
5. Erstelle den Type Alias `EmployeeRecord`, der ein Objekt `{ id: number, data: Employee }` definiert.
6. Erstelle die Klasse `Company` mit einer Index-Signatur, die `EmployeeRecords` speichert, wobei `id` als Schlüssel dient. Füge die Methoden `addEmployee(employee: Employee)` und `getEmployee(id: number)` hinzu.