thecodecampus</>

TypeScript

für Java- und .net-Entwickler

Über uns

theCodeCampus.de - Weiter.Entwickeln.

<> Gegründet 2013
Schulungen (seit 2007)
Projekt-Kickoffs
Unterstützung im Projekt </>

w11k GmbH - the Web Engineers

<> Gegründet 2000
Entwicklung / Consulting
Web / Java
Esslingen / Siegburg </>

Verlosung

TWEET AND LET RETWEET

1x **Angular Schulung** im Wert von 1.450,- €

5x **Geek & Poke Bücher**

Teilnahme via Twitter

Dein Text + „@theCodeCampus #angular Schulung #herbstcampus16“
Tweet mit meisten Retweets gewinnt

Teilnahme bis 05.09.16 10:00
Bekanntgabe der Gewinner via Twitter

Über mich

Jan Blankenhorn

<> Software-Entwickler, Geschäftsführer

Java Entwickler seit 10 Jahren

Web seit 5 Jahren

AngularJS, TypeScript, ...

blankenhorn@w11k.de </>

Umfrage

<> Java Entwickler?

<> C# Entwickler?

<> JS Entwickler?

<> CoffeeScript Entwickler?

<> TypeScript Entwickler?

FuckJS

Warum JS für uns OOP Entwickler so schlimm ist ..

JS / ECMA Script Versionen

<> Verschiedene Broser unterstützen verschiedene JavaScript Features
- http://kangax.github.io/compat-table/es6/

<> ES5: (2011): Aktuell verbreiteter Standard, den quasi alle Browser unterstützen

<> ES6 / ES2015 (2015): Unterstützung durch mittlerweile viele Browser

<> ES2016 ...

<> **Beispiele hier: ES5**

Schwach und dynamisch typisiert

<> **Wie streng sind die Typen?**
- **schwache** Typisierung: `s = "Zahl " + 5;`
- **starke** Typisierung: `s = "Zahl " + str(5)`

<> **Wann wird der Typ geprüft?**
- **statische** Typisierung: `String s = new String();`
- **dynamische** Typisierung: `var s = "a"; s = 3;`

<> **Fehler werden zum falschen Zeitpunkt gefunden**
- Zur Laufzeit
- Abhilfe: Tests, Linter (sehr statisch, falsche Sicherheit)
- Nervig: Zum Entwicklungszeitpunkt sind Datentypen nur über Code, oder Netzwerktab ersichtlich

<> `LIVE CODING!`

Scopes

<> Kein Block Scoping (Im Gegensatz zu Java und C#)

<> Global / Function Scope

<> Hoisting

<> `LIVE CODING!`

this

<> this in Java / C# => Aktuelles Objekt

<> In JS ist dies leider nicht so ...

<> Kontext von this bestimmt der Aufrufer

<> `var self = this;`

<> `LIVE CODING!`

Prototypische Vererbung

<> Wer verwendet Klassenorientierte Vererbung?

<> Schon mal von Prototypischer Vererbung gehört?

<> JS: Da keine Klassen, werden Objekte vererbt

<> Zur Laufzeit Änderungen möglich, d.h. der Prototyp kann sich ändern

<> Lesend: Zugriff an Prototyp Eigenschaften

<> Schreibend: Dann im eigenen Objekt

<> `LIVE CODING!`

Am allerschlimmsten aber …

<> Datentypen zum Zeitpunkt der Entwicklung noch klar

<> Später, aber
- viel Arbeit mit dem Netzwerk Tab

<> Code von vor 2 Monaten

<> Große Projekte (Code von anderen)

<> Refactorring

TypeScript

In 1 Minute

JS / ECMA Script Versionen

<> **Modernes EcmaScript - Neues schon jetzt nutzen**
- + statische Typisierung
- Syntax ist kompatibel zu JavaScript
- Ziel: Möglichst gute Interopabilität mit bestehendem JS Code

<> **Angular2 wird in TS entwickelt (AtScript Story)**

<> **Microsoft**
- Open Source
- Git Hub
- Apache License

Setup

<> `node --version`

<> `npm install typescript --save-dev`

<> `./node_modules/.bin/tsc --version`

<> `tsconfig.json`

<> **Alternative: Built-In Compiler in IntelliJ**

<> `LIVE CODING!`

Was ist ähnlich

zu Java und C#

Statische Typisierung

<> **Datentypen:** String, Number, Boolean, Null, Undefined, Enum

<> Eigene Typen und Klassen können erstellt werden

<> Was passiert zur Laufzeit??

<> `LIVE CODING!`

Arrow Functions

<> Java/C# „Lambda Expression“

<> Hauptfeature: this Capturing

<> `() => {}` anstatt `function() {}`

<> `LIVE CODING!`

Klassen und Vererbung

<> Sehr ähnlich zu Java und C#

<> Klassen und Interfaces

<> Konstruktor Parameter haben spezielle Bedeutung

<> Keine Mehrfachvererbung

<> `LIVE CODING!`

Generics

<> Ähnlich zu Java und C#

<> Verwendung in Klassen, Funktionen etc ..

<> Keine Varianz, aber `extends` ist möglich

<> `LIVE CODING!`

Was ist anders

als in Java und C#

Typsystem

<> **JavaScript: Structural Typing**
- Objekte sind kompatibel, wenn deren Struktur gleich ist
- Egal, ob Objekt durch eine Klasse, oder durch Objektliteral erzeugt wurde

<> **Java / C# vs. Nominal Typing: Der Name entscheidet**

<> **Type Inferencing: (Gibt es in C#, in Java nur als Diamond Operator)**

<> **Any Type: Variable können Werte jedes Typs zugewiesen werden, dynamisch veränderbar**

JavaScript kennt kein Überladen von Methoden

<> **Geht in JS nicht, 2. Methode überschreibt erste**
- `arguments`

<> **In TS daher nicht erlaubt**
- Da Typen zur Laufzeit wegfallen
- Dafür -> Rest Parameters in TS == Arguments

<> `LIVE CODING!`

Dekoratoren (Annotations)

<> Dekoratoren sehen ähnlich aus wie Annotations, aber:

<> In Java: Statische Meta-Daten

<> In TypeScript: Funktionsaufrufe

<> `tsconfig "experimentalDecorators": true`

<> `LIVE CODING!`

Modulsystem

<> So richtig hat Java keins. OSGi natürlich, bald Jigsaw. C#??

<> JS kennt veschiedene Modulsysteme mit versch. Syntax und Features

<> Verwendung in TypeScript
- Syntax = ECMASCript 6
- Kann in versch. Modulsysteme generiert werden
- SystemJS, CommonJS, ...
- `tsconfig.json`
- Ohne Export / Import, ist eine Datei nur 1 Script

<> `LIVE CODING!`

Namespaces und Typings

<> **Namespaces**

- Ähnlich zu Java: Packages -> Eindeutige Namen!
- Motivation: JavaScript Bibliotheken wie z.b. `Lodash` abbilden
- `_.forEach()`
- Modulsystem sind aber besser

<> **Typings**

- Problem: JS Bibliotheken haben keine Typinformationen
- Ähnlich zu C Header Dateien `.d.ts`
- Schlüsselwort `declare`
- Typings Projekt https://github.com/typings/registry

<> `LIVE CODING!`

Weitere Typen

<> Union Types

<> TypeGuards

<> Smart Cast

<> ...

<> `LIVE CODING!`

Warum wir TS lieben

Integration in JS sehr gut gelungen

- Koexistens JS und TS Code funktioniert sehr gut
- Schrittweise Migration möglich
- Gute Verknüpfung von OO und Js Welt

Wartbarkeit von Projekten

<> Statische Typisierung sorgt für deutlich bessere Wartbarkeit

<> Vor allem in größeren Projekten

<> Mehrere Teams

Arbeit in der IDE

<> Code Completion

<> Refactoring

<> Quellcode Navigation

<> Die IDE's sind schon ganz gut, werden immer besser

Hinter dem Vorhang

Compiler

<> **Compiler ist in TypeScript geschrieben, kann auch in Browsern verwendet werden**

<> **TSserver**

- Kann von den IDE's angesprochen werden, für jegliche Information (z.B. CodeCompletion)
- IDE's müssen Compiler nicht selbst entwickeln

Sourcemaps

<> Im Browser läuft JS

<> Via Sourcemaps Debugging des TS Codes (auch in der IDE) möglich

info@thecodecampus.de
@thecodecampus

www.w11k.de
www.thecodecampus.de