The Powerful Three

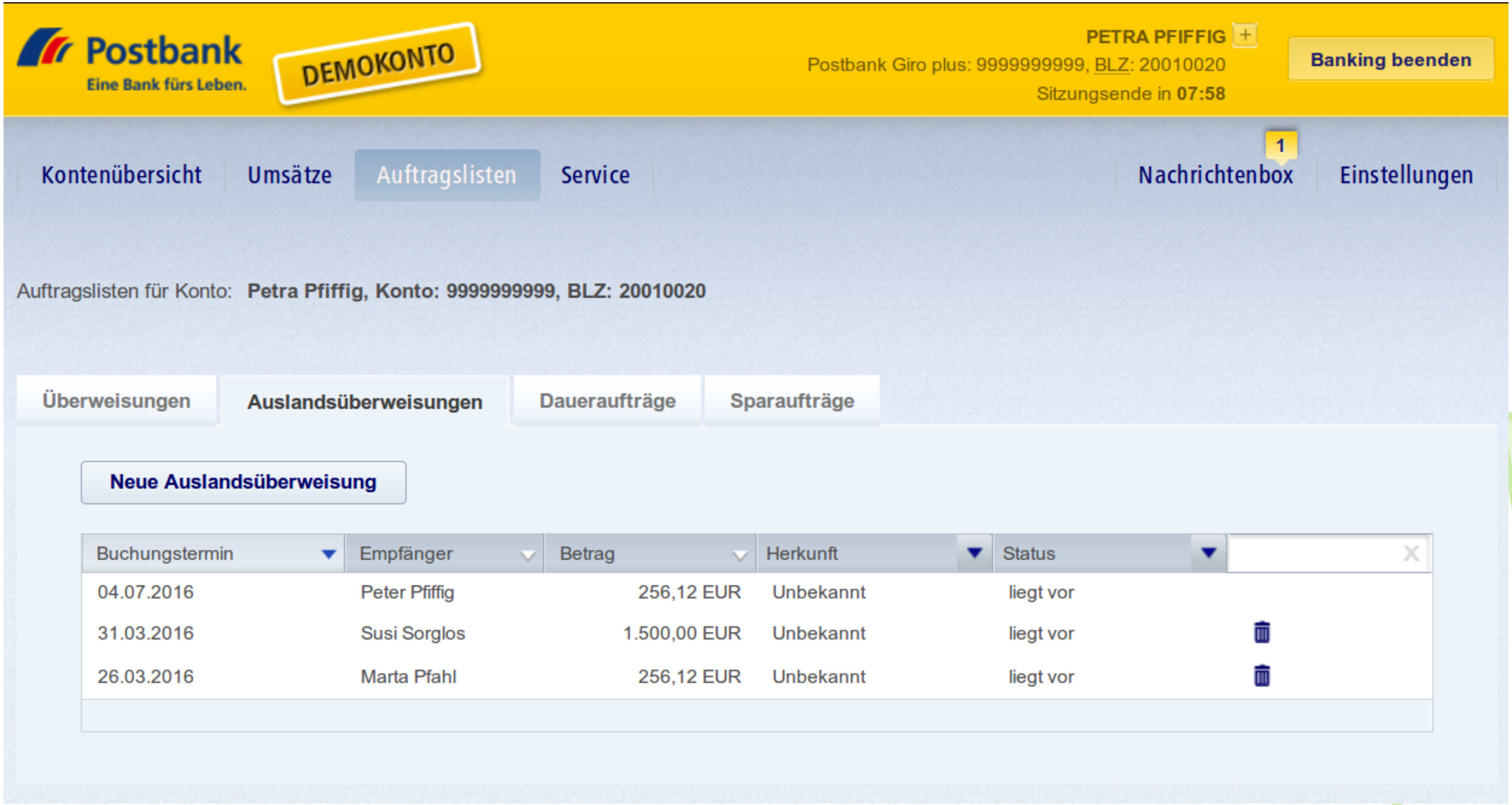
NetBeans IDE, TypeScript, Angular 2

KARSTEN
SITTERBERG

@kakulty

- Freelance trainer/consultant since 2013
- Working for trion especially for frontend topics
- Co-founder of the JUG Münster

Web applications

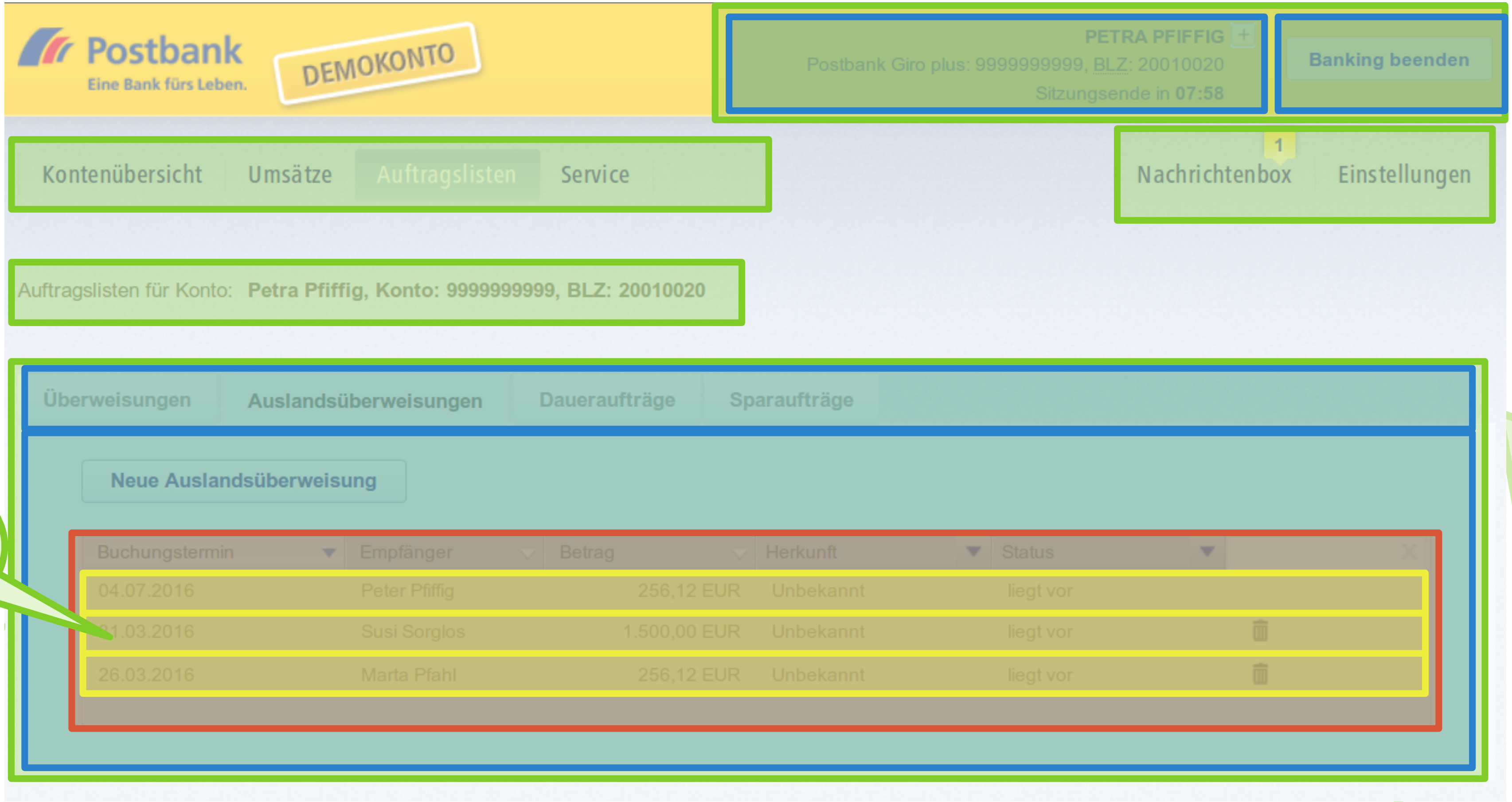


PETRA PFIFFIG +
Postbank Giro plus: 9999999999, BLZ: 20010020
Sitzungsende in **07:58**

Auftragslisten für Konto: **Petra Pfiffig, Konto: 9999999999, BLZ: 20010020**

Buchungstermin	Empfänger	Betrag	Herkunft	Status	X
04.07.2016	Peter Pfiffig	256,12 EUR	Unbekannt	liegt vor	
31.03.2016	Susi Sorglos	1.500,00 EUR	Unbekannt	liegt vor	🗑
26.03.2016	Marta Pfahl	256,12 EUR	Unbekannt	liegt vor	🗑

Components of web applications



PETRA PFIFFIG

Postbank Giro plus: 9999999999, BLZ: 20010020

Sitzungsende in **07:58**

Auftragslisten für Konto: **Petra Pfiffig, Konto: 9999999999, BLZ: 20010020**

Buchungstermin	Empfänger	Betrag	Herkunft	Status
04.07.2016	Peter Pfiffig	256,12 EUR	Unbekannt	liegt vor
1.03.2016	Susi Sorglos	1.500,00 EUR	Unbekannt	liegt vor
26.03.2016	Marta Pfahl	256,12 EUR	Unbekannt	liegt vor

Reuse

Components

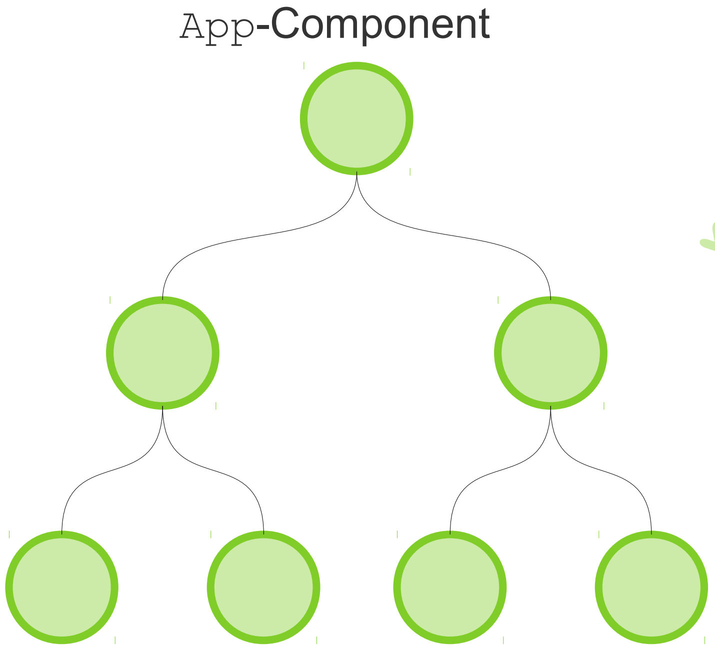
Component Stacking

Components in general

- Provide level of reuse
- Delegation and decoration
- **Single** responsibility of each component

Separation of Concerns

An app is just a tree of components!



Custom components in HTML

```
<postbank-banking></postbank-banking>
```

```
<postbank-page>
    <postbank-header></postbank-header>
    <postbank-app></postbank-app>
</postbank-page>
```

```
...
<transfer-menu></transfer-menu>
<transfer-tab>
    <new-transfer-button></new-transfer-button>
    <transfers-table></transfers-table>
</transfer-tab>
...
```

Components in web applications

- Improve developer productivity
  - Easier to write, reduced noise
  - Better way of re-use than copy-pasta
- Well established on server side, f.e. JSF/Apache Wicket
- Client side (HTML/JS) applications on the rise
  - Past: HTML/JS + jquery, no components
- **Current approach**:
  - **Single page applications (SPA's) –** client side HTML/JS-Applications
  - **Server** provides persistence and business transactions

Single Page Applications

- Technology

- Custom HTML tags
  - Web Components

- SPA-Frameworks

  → AngularJS, Angular 2

Single Page Applications

- Mobile first
- Responsive
- Also Offline First

- One HTML Document (Single Page)
- **Contents dynamically** loaded & interchanged
  - Content-Updates small (compared to overall new Page)

  → Less Network Traffic / Loading Times

  → Better User Experience
- More Modular (Components)

  → Loose Coupling ↔ Separation of Concern

Angular 2

- Angular 2 RC 1 is there!
- **Multilingual** Development
  - ES5, ES2015, **TypeScript**, Dart, …
  - Major Part of Documentation is TypeScript
  - Angular 2 itself developed in TypeScript
- Modular Applications
  - Dependency Injection

  → Testability
- Event-Based Change Detection

Angular 2

- **Reactive** Change Detection: Built for **Speed**
- **Components: Simple**/Productive
- URL Routing
- Legacy Browser Support (down to IE9)
- Templates for DOM-Changes
- Shadow DOM for Encapsulation

WebComponents

- Optional rendering without DOM access
  - Server side rendering (SEO, isomorphic applications)
  - WebWorker for rendering offload

Angular 2 Under the Hood

Instead of reinventing the wheel:

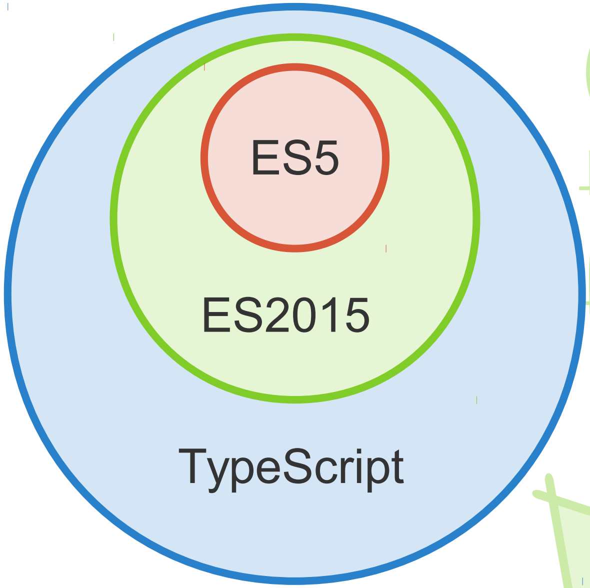
- SystemJS (Module loader)
- ES6-shim (Browsers without ES6 support)
- **RxJS** (Reactive programming)
  - Observables for **asynchronous**, event-based program flows
- **zone.js**
  - Execution **contexts** for asynchronous tasks
- reflect-**metadata**
  - Dependency injection via **decorators**

TypeScript

- TypeScript extends ES2015 extends ES5 functionality
- Static Typing
  - Compile-Time Security
  - Clearer Docs/Tooling/Refactoring
- Visibility, private/public
- Interfaces, Classes, Inheritance

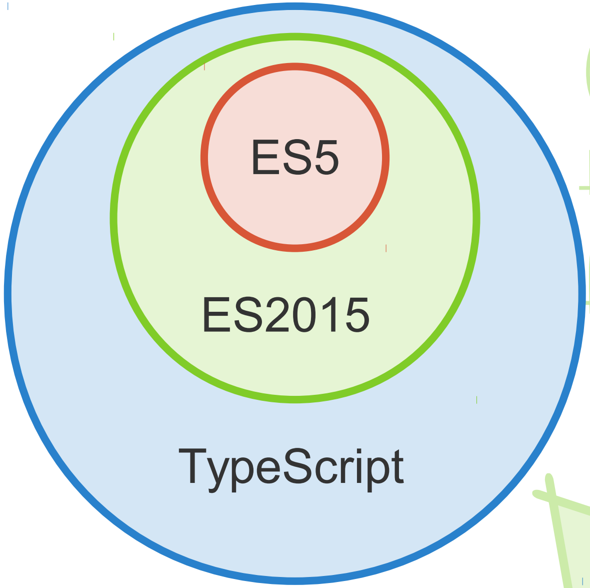
Compile-Time

→ IDE can do static analysis and Code-Completion



TypeScript

- TypeScript extends ES2015 extends ES5 functionality
- Browser Support only for ES5
  - Even lacking ES2015 support
  - ES5 as “Web-Assembler”
- Compilation from TypeScript to ES5:
  - Often called “Transpilation”
  - TypeScript Compiler: `tsc`



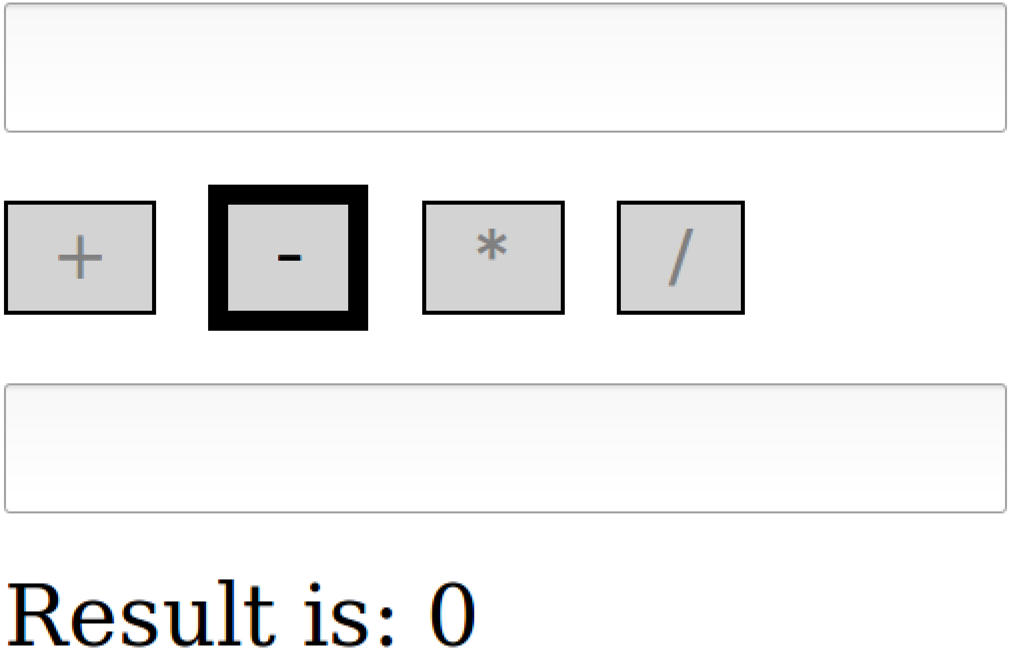
TypeScript – Example

```
interface Person {
    firstName: string;
    lastName: string;
}
function greet(person : Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}
```

TypeScript – Example

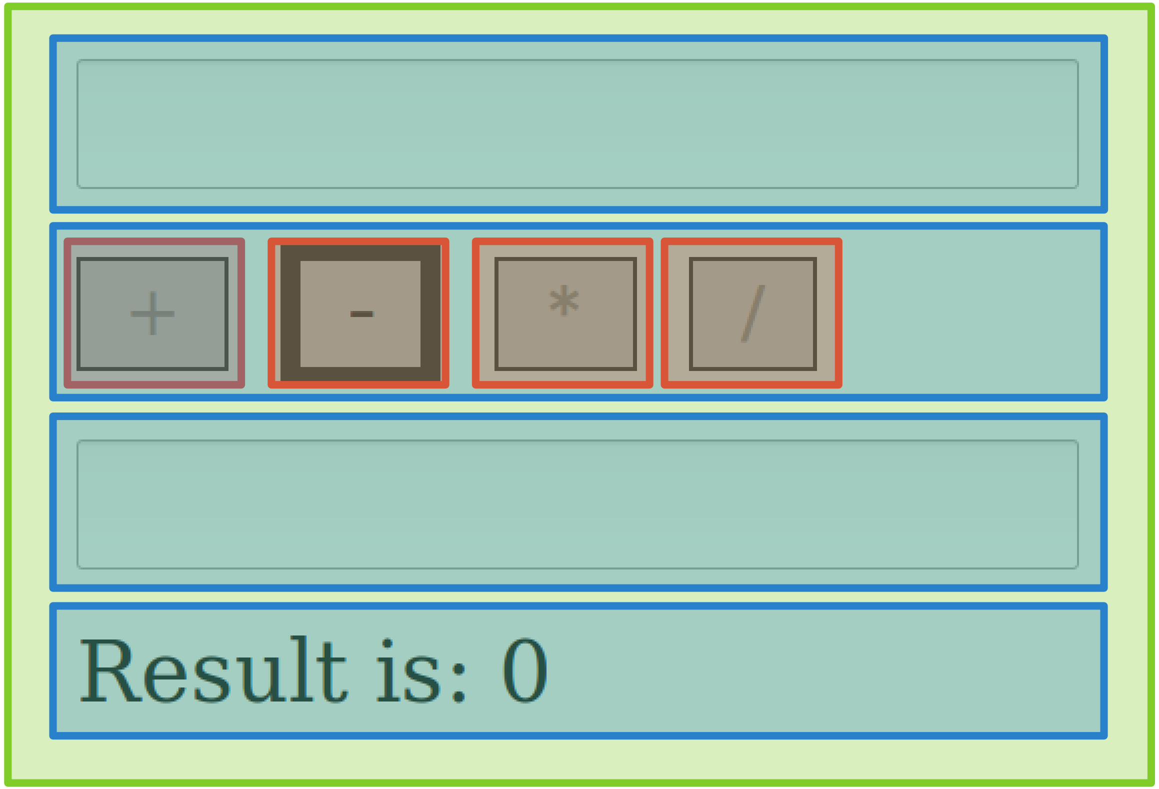
```
interface Person {
    firstName: string;
    lastName: string;
}
function greet(person : Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}


class Student implements Person {
    constructor(public firstName, public lastName, public course) {
    }
}
var user = new Student("Jane", "User", "Biology");
document.body.innerHTML += greet(user);
```

Angular 2 – Example: Calculator



Result is: 0

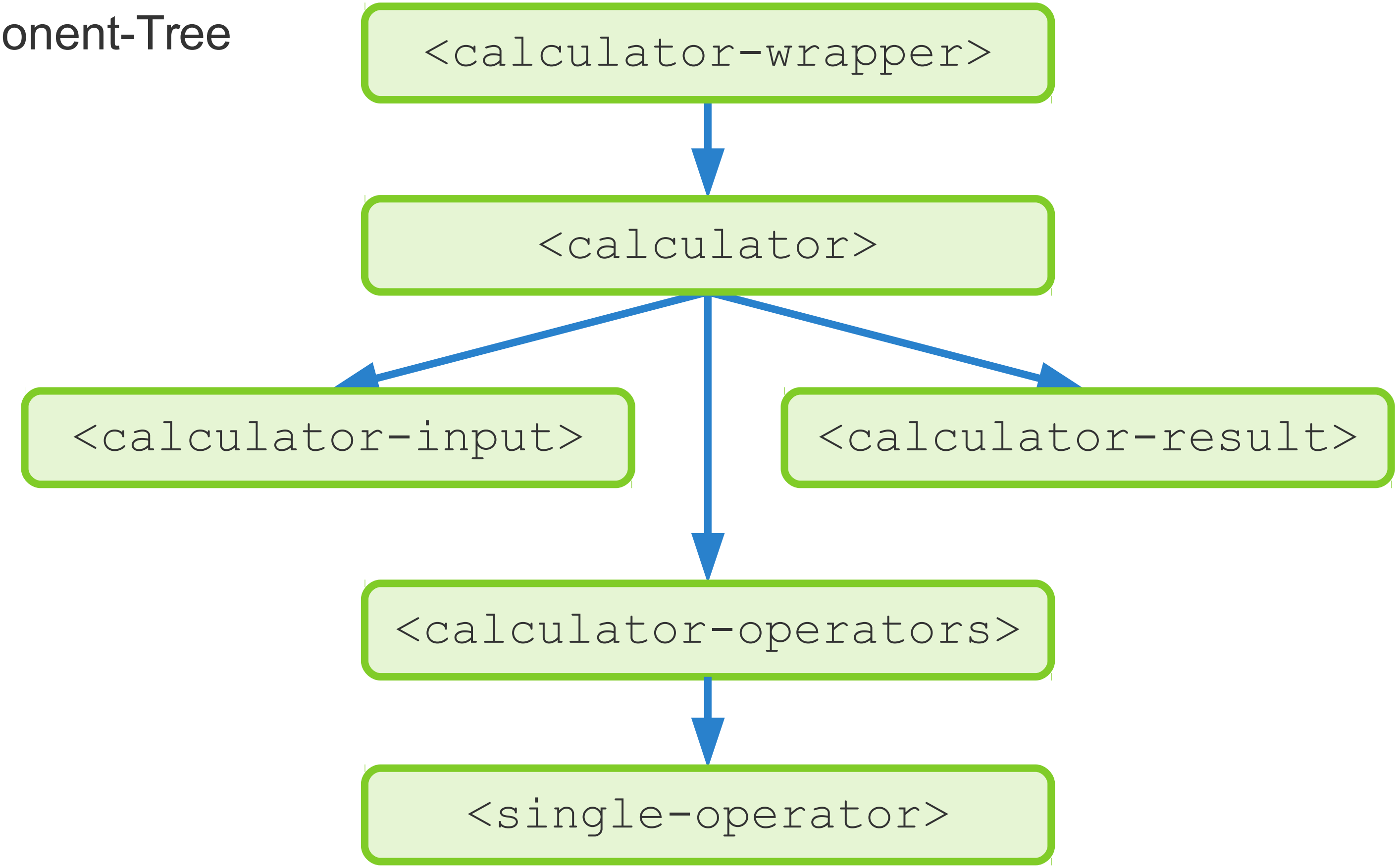
Angular 2 – Example: Calculator



Result is: 0

Angular 2 – Component

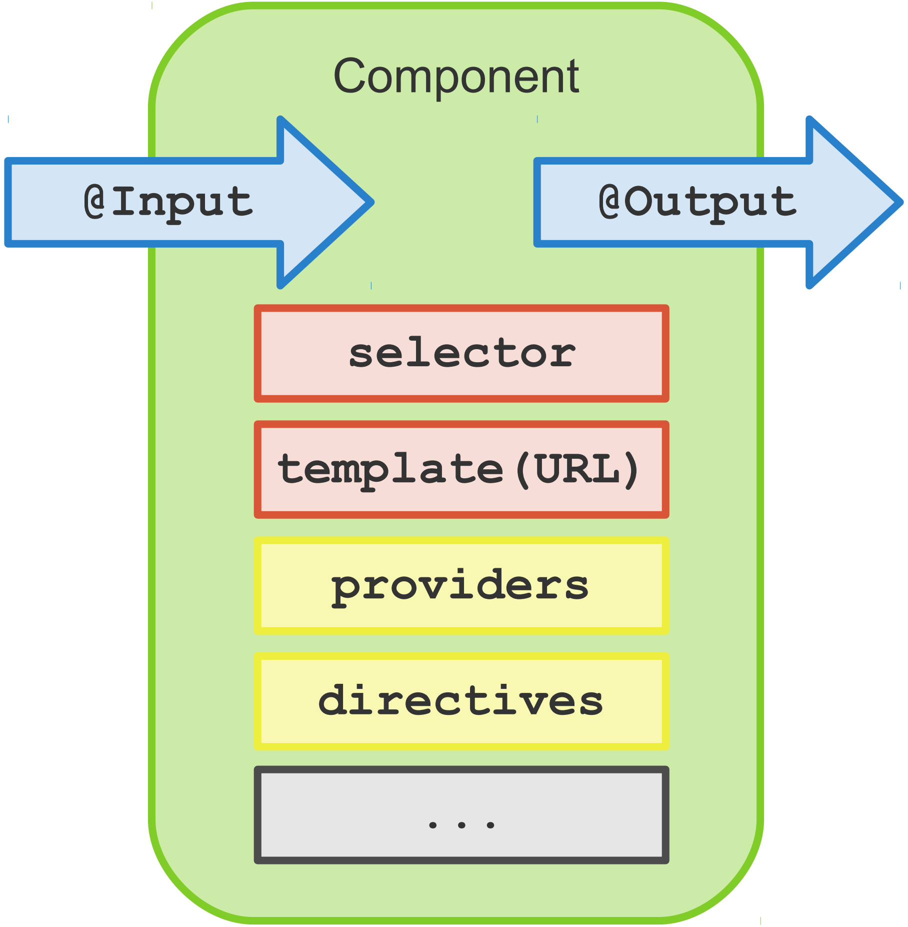
- Component-Tree



Angular 2 – Component

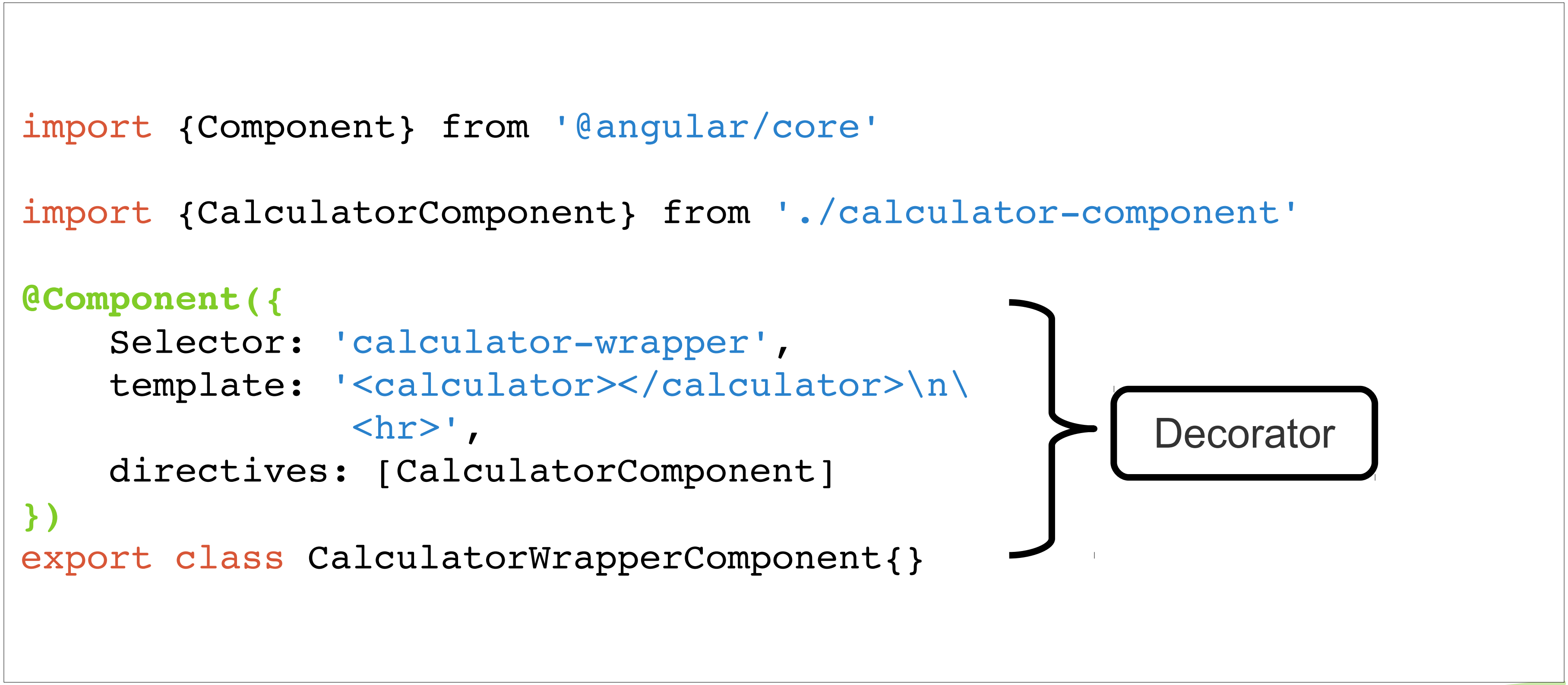
- Acts as Container for View
- Communicates with other Components
- Delegate Logic to Services/ Fetch Data from Services

Services:
- View-Specific Logic
- Communication to Back-End



Angular 2 – Component

- Basic Blueprint: `CalculatorWrapperComponent.ts`



```
import {Component} from '@angular/core'

import {CalculatorComponent} from './calculator-component'

@Component({
    Selector: 'calculator-wrapper',
    template: '<calculator></calculator>\n\
               <hr>',
    directives: [CalculatorComponent]
})
export class CalculatorWrapperComponent{}
```

Angular 2 – Component

- Basic Blueprint: `CalculatorComponent.ts`

```
import {Component} from '@angular/core';

import {OperationsComponent} from './operations-component';
import {InputComponent} from './input-component';
import {ResultComponent} from './result-component';
import {CalculatorService} from './calculator-service';

@Component({
    selector: 'calculator',
    template: `... `,
    directives: [OperationsComponent, InputComponent, ResultComponent]
})
export class CalculatorComponent {
    \\...
}
```

Angular 2 – Component

- Basic Blueprint: `SingleOperatorComponent.ts`

```
import {Component, Input, Output, EventEmitter} from '@angular/core';

@Component({
    selector: 'single-operator',
    template: `<button [class.active]='selected'
               (click)='changeOperation()'> {{operation}} </button>`
})
export class SingleOperatorComponent {
    @Input() selected: boolean;
    @Output() operationChanged = new EventEmitter();
    private operation: string = '+';
    private changeOperation() {
        this.operationChanged.emit(this.operation);
    }
}
```

Angular 2 – Template Syntax

```
<single-operator (operationChanged)="operationChange(event)"
    [selected]="isSelected()" >
</single-operator>
```

@Input()
Property Binding

@Output()
Event Binding

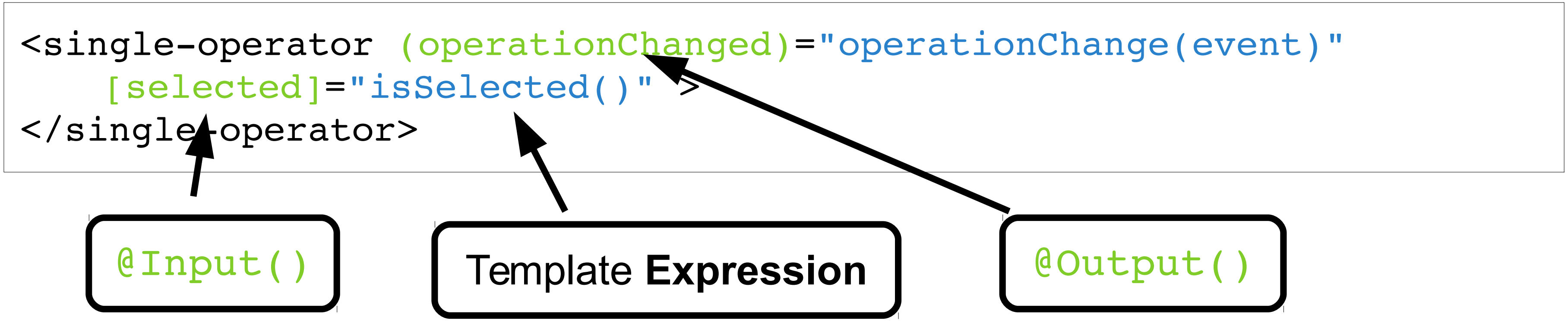
- **One-Way Binding:**
  - Possible Input-Bindings:
    DOM-Properties, CSS-Classes, Styles, HTML-Attributes

- **Two-Way Binding** also possible

```
      <input [(ngModel)]="todo.text"></input>
```

Banana in a Box
[()]

Angular 2 – Template Syntax



```
<single-operator (operationChanged)="operationChange(event)"
    [selected]="isSelected()" >
</single-operator>
```

@Input()

Template **Expression**

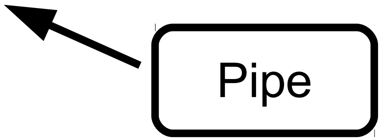
@Output()

- Expressions should
  - Have **no** (visible) **side effect**
  - Execute **quick**
  - Be **simple**
  - Be **idempotent**

Angular 2 – Template Syntax

- Pipes

```
<div>{{ title | uppercase }}</div>
```



Pipe

- “Elvis”-Operator `?.`

```
{{currentHero?.enabled}}
```

↔

```
{{currentHero && currentHero.enabled}}
```

Angular 2 – Service

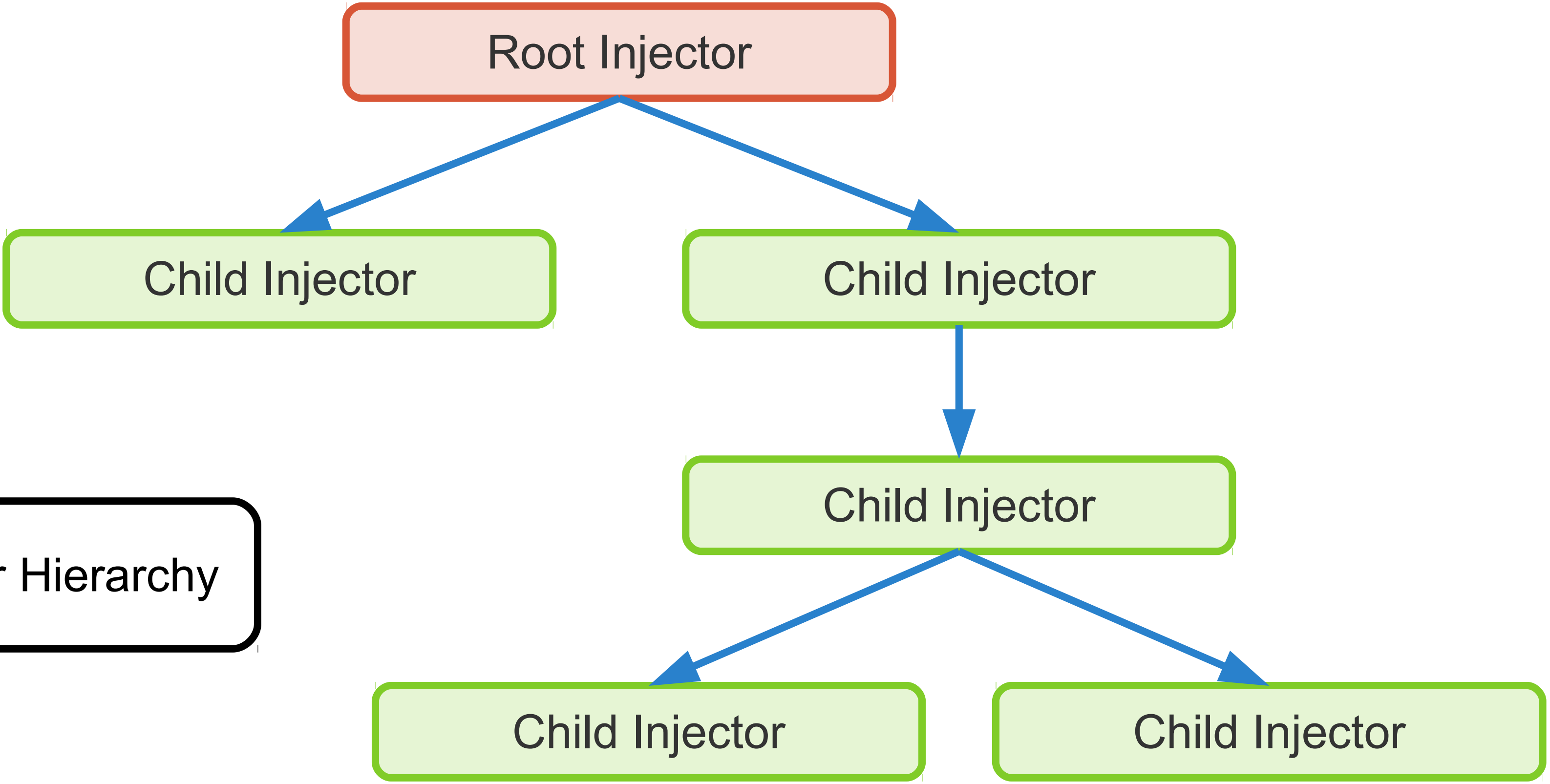
- Basic Blueprint

```
import {Injectable} from '@angular/core';

@Injectable()
export class CalculatorService {}
```

- Inject the Service into the Component

```
import {Component} from '@angular/core';
import {CalculatorService} from './calculator.service';

@Component({    ...
    providers: [CalculatorService]
})
export class CalculatorComponent {    ...
    constructor(private calculatorService: CalculatorService) { }
}
```

Angular 2 – Dependency Injection



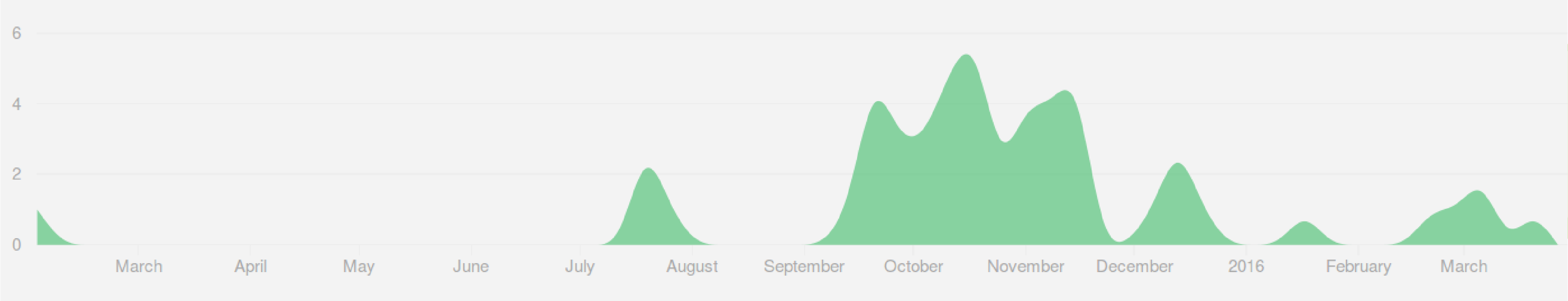
Injector Hierarchy

NetBeans IDE

- Free, Open Source IDE
- Web Development Capabilities:
  - HTML/CSS/JS
  - AngularJS (Angular 1) Support
  - PHP
- Functionality extensible via Plugins
  - ~200 official Plugins
  - Even more available from unofficial Sources
  - Develop your **own Plugin**!

NetBeans – TypeScript Support

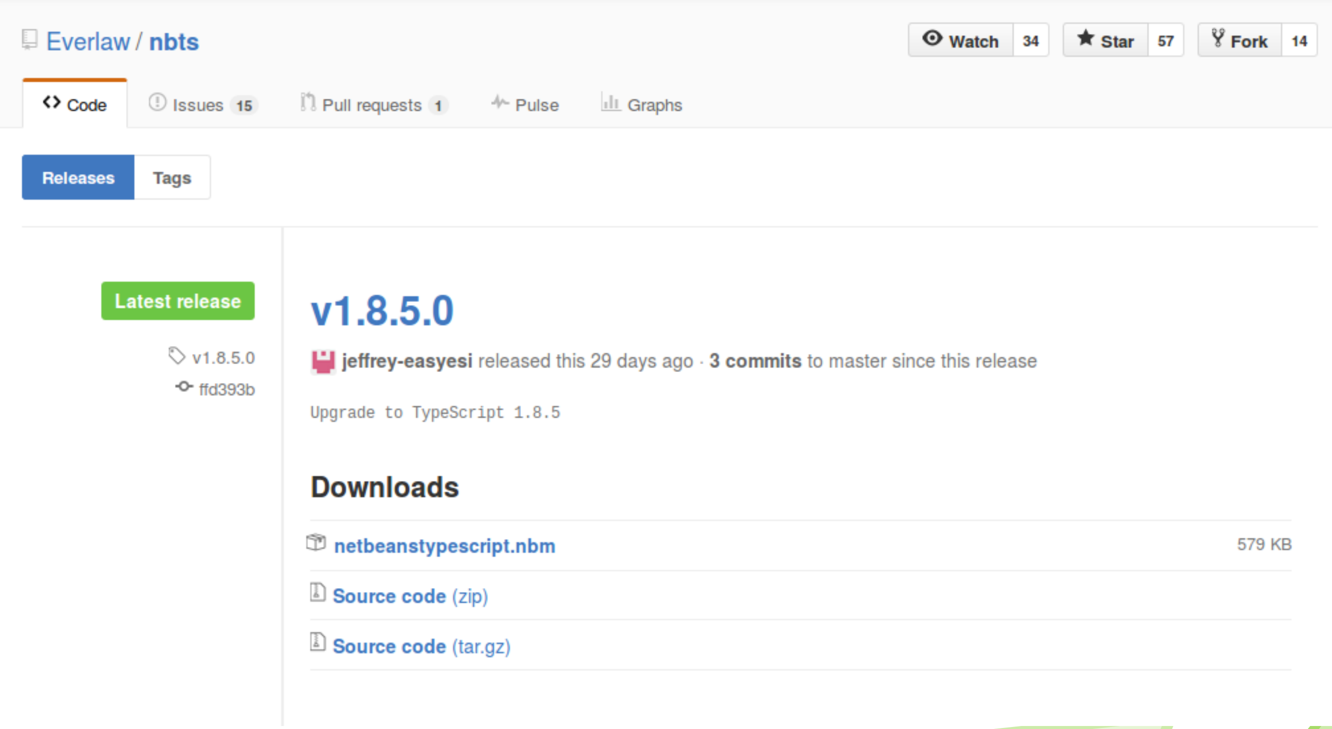
- GeertjanWielenga/TypeScript
  - 26 Sep. 2015 – 01 Okt. 2015
- Everlaw nbts
  - (01 Feb. 2015) 29 Sep. 2015 – 20 Mar. 2016



Everlaw nbts-Plugin

- Community Driven Plugin
- Actively Developed
- Multiple Contributors
- Collaboration via GitHub

https://github.com/Everlaw/nbts



v1.8.5.0

jeffrey-easyesi released this 29 days ago · 3 commits to master since this release

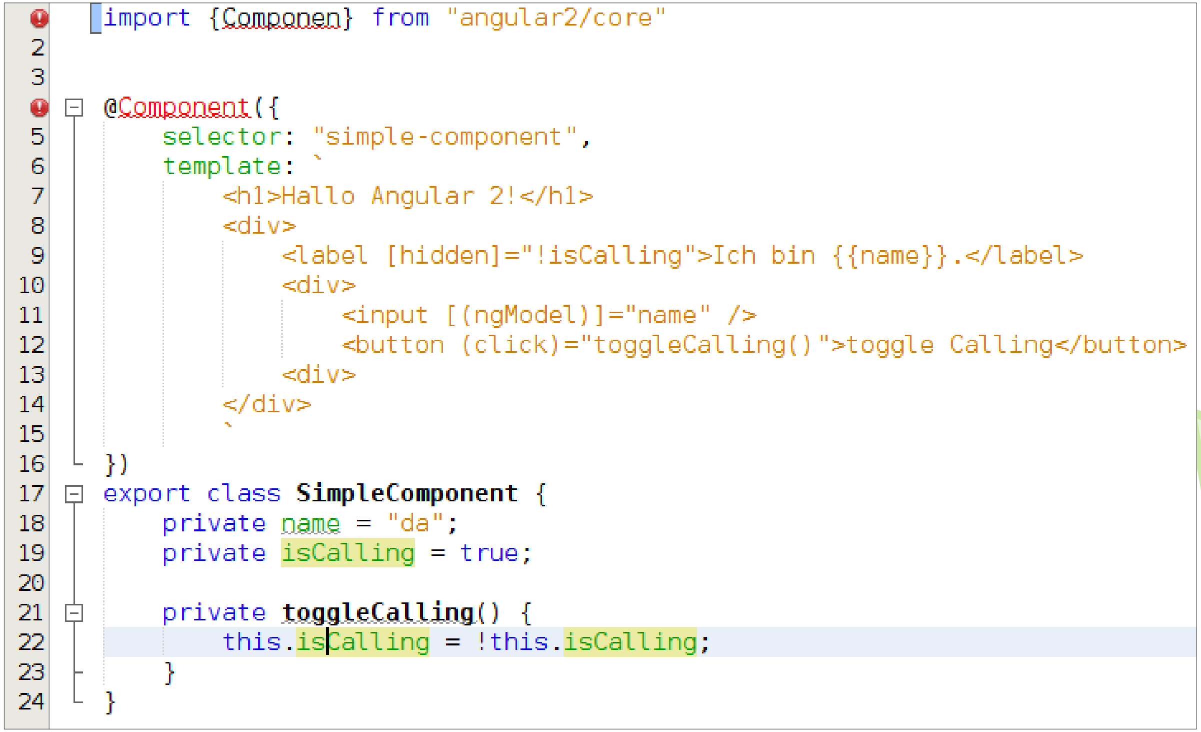
```
Upgrade to TypeScript 1.8.5
```

Downloads

netbeanstypescript.nbm	579 KB
Source code (zip)	
Source code (tar.gz)	

Everlaw Plugin Features

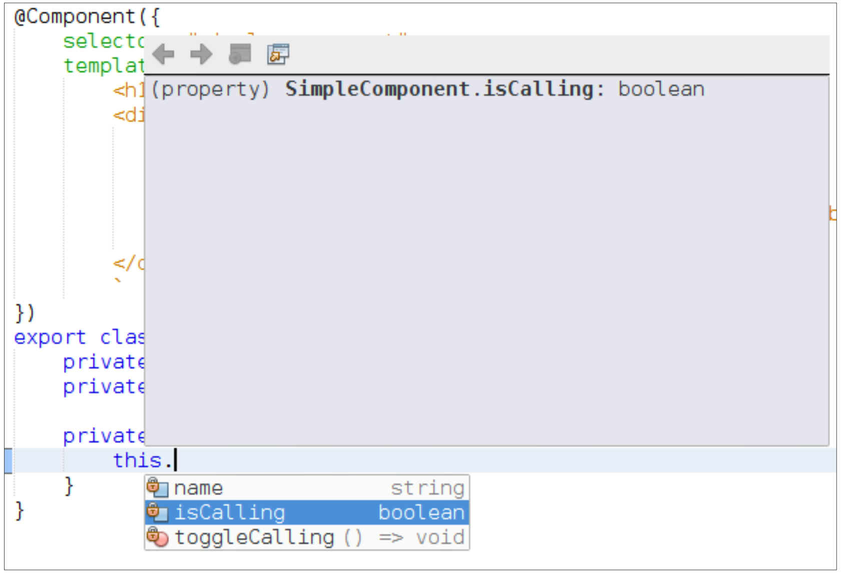
- Syntax highlighting
- Error Checking
- Usages Highlighting
- Basic Refactoring



```
import {Componen} from "angular2/core"


@Component({
    selector: "simple-component",
    template: `
        <h1>Hallo Angular 2!</h1>
        <div>
            <label [hidden]="!isCalling">Ich bin {{name}}.</label>
            <div>
                <input [(ngModel)]="name" />
                <button (click)="toggleCalling()">toggle Calling</button>
            <div>
        </div>
        `
})
export class SimpleComponent {
    private name = "da";
    private isCalling = true;

    private toggleCalling() {
        this.isCalling = !this.isCalling;
    }
}
```

Everlaw Plugin Features

- Code Completion

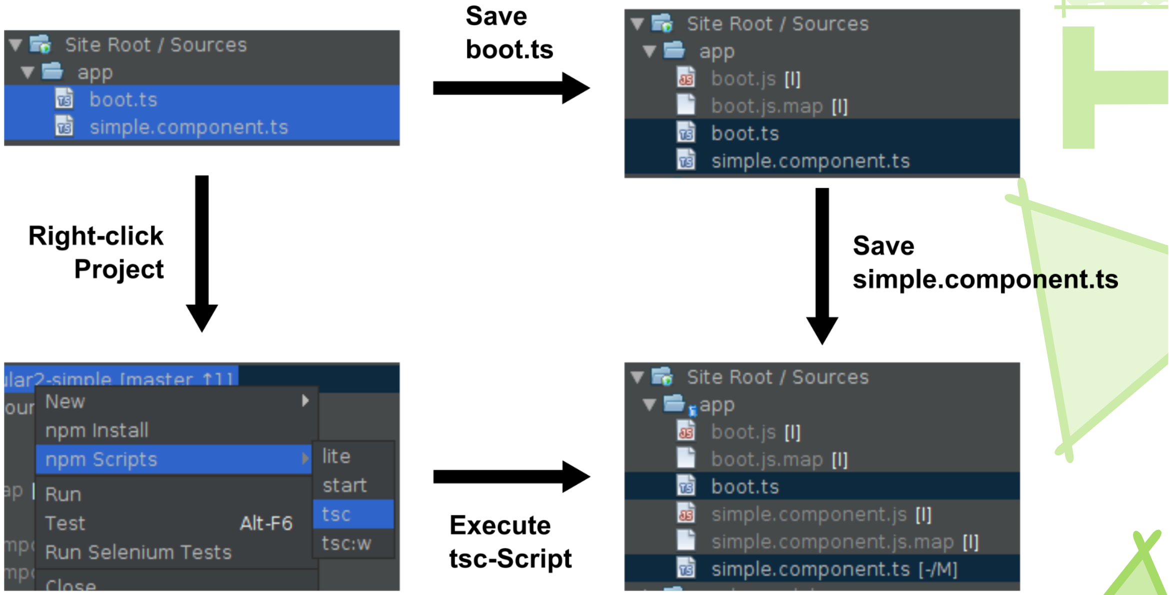


```
@Component({
    selecto
    templat
        <h1
        <di



        </c
})
export clas
    private
    private

    private
        this.
    }
}
```

(property) **SimpleComponent.isCalling**: boolean

name	string
isCalling	boolean
toggleCalling ()	=> void

Compile-on-Save



Save boot.ts

Right-click Project

Save simple.component.ts

Execute tsc-Script

Missing Features for Angular 2

- Template language support
- Debugger Support (Breakpoints, Variable inspection ...)
- Custom-component detection/support
- Extract functions-refactoring not available
- Refactoring of filenames leaves orphans
  - Workaround: Special build folder for ES5 artifacts
- tsconfig.json editing-support

Demo

Outlook: Alternatives to TypeScript

- ES6/ES2015 NetBeans support in development
  - Planned for 8.2
  - Transpilation to ES5 required (browser support)
- Dart: Not supported in NetBeans
  - Small ecosystem
  - Probably replaced by TypeScript or ES2017
- ES5
  - No transpilation
  - Language supported, but no special Angular 2 support

Interested in more?

We provide you with training, consulting, development

Java, Spring, AngularJS and Angular 2

Contact us: www.trion.de

Thank you
for your attention

I am looking forward to answer your questions at
karsten.sitterberg@trion.de