Spotkania z TypeScript

Wykład 2:

TypeScript

...czyli jak zaprzyjaźnić programistów .NET z JavaScriptem

Statyczne typowanie

```
function validate(field) {
    var email = field.value;
    var atpos = email.indexOf("@");
    var dotpos = email.lastIndexOf(".");

    return atpos >= 1 && (dotpos - atpos >= 2);
}
```

Statyczne typowanie

```
function validate(field: HTMLInputElement): boolean {
    var email: string  = field.value;
    var atpos: number  = email.indexOf("@");
    var dotpos: number = email.lastIndexOf(".");

    return atpos >= 1 && (dotpos - atpos >= 2);
}
```

Statyczne typowanie

Zalety statycznego typowania:

- Mniejsze szanse na błąd wynikający z literówki bądź niedozwolonego wykorzystania funkcji
- Automatyczne podpowiedzi (np. dla atrybutów obiektów)
- Czytelniejszy, samodokumentujący się kod

Typy podstawowe

- Typy proste

```
number    boolean    string
```

- Tablice

```
var arr: number[];
var arr: Array<number>;
```

- Typ „any“

```
var obj: any;
```

- Unia (alternatywa typów)

```
var x: number | boolean | string ;
```

Typ „any”

- „Wiem co robię, zaufaj mi!“
- Oznacza, że obiekt może być dowolnego typu (wyłącza kontrolę nad typem)
- Zapobiega odgadywaniu typu na podstawie użycia (brak deklaracji typu wcale nie oznacza, że zostaje domyślnie przyjęty typ „any“)
- Często wykorzystywany w przypadku obiektów dla których brak prawidłowej deklaracji typu

Odgadywanie typu

```
var sqr = function (a: number) { return a * a; }
var val = sqr(4);
```

var val: number

```
var tmp_value = "string";
tmp_value = 5;
```

var tmp_value: string

Type 'number' is not assignable to type 'string'.

Argumenty funkcji

```
// Argumenty opcjonalne i domyślne
function foo(x?: number, y: number = 5) {
    return x ? x / y : 0;
}

// Funkcja wieloargumentowa
function buildName(firstName: string, ...restOfName: string[]) {
    return firstName + " " + restOfName.join(" ");
}

var employeeName = buildName("Joseph", "Samuel", "Lucas", "MacKinzie");
```

Klasy i interfejsy

… czyli bardziej intuicyjny OOP w TypeScript

Klasy i interfejsy

```
1  class Greeter {
2      private element: HTMLElement;
3      public span: HTMLElement;
4      timerToken: number;                  // defaultowo jest public
5
6      constructor(element: HTMLElement) {
7          this.element = element;
8          /*...*/
9      }
10
11     private internalMethod() { }
12     start() { /*...*/ }
13     stop() { /*...*/ }
14 }
```

Klasy i interfejsy

```
1  var Greeter = (function () {
2      function Greeter(element) {
3          this.element = element;
4          /*...*/
5      }
6      Greeter.prototype.internalMethod = function () { };
7      Greeter.prototype.start = function () { };
8      Greeter.prototype.stop = function () { };
9      return Greeter;
10 }());
11
```

Po kompilacji do JavaScript tracona jest informacja o typie i enkapsulacja

Klasy i interfejsy

Klasy w TypeScript

Polimorfizm i dziedziczenie

Interfejsy

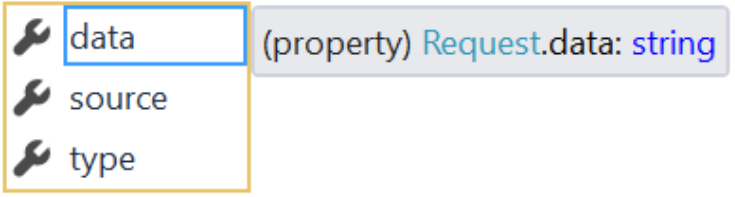
- Deklarują pewien interfejs dla danego bytu, np. klasa zadeklarowana jako implementująca dany interfejs, musi go faktycznie dostarczać
- Mechanizm czasu kompilacji (interfejsy znikają z kodu wynikowego)
- Mogą deklarować atrybuty i metody opcjonalne

Interfejsy

```
interface Clickable {
    clickCounter: number;
    unclickCounter?: number;

    click();         // metoda wymagana
    unclick?();      // metoda opcjonalna
}

class GUIObject implements Clickable {
    clickCounter: number;

    public click() { }
}
```

Interfejsy (nie tylko dla klas)

```typescript
interface Request {
    type: string;
    source: string;
    data?: string;
}

function createInitRequest(): Request {
    return {
        type: "init",
        source: "system"
    }
}

var r = createInitRequest();
r.
```



Moduły i zewnętrzne biblioteki

Moduły i zewnętrzne biblioteki

Wewnętrzna organizacja kodu i zewnętrzne zależności

Moduły i zewnętrzne biblioteki

```
/**
 * Wymagany moduł jest w innym pliku
 * Musimy poinformować kompilator, by najpierw skompilował zależność
 **/

/// <reference path="Validation.ts" />
module Validation {
    // Wewnętrzna zmienna
    var lettersRegexp = /^[A-Za-z]+$/;
    // Klasa wystawiona na zewnątrz modułu (export)
    export class LettersOnlyValidator implements StringValidator {
        isAcceptable(s: string) {
            return lettersRegexp.test(s);
        }
    }
}
```

Moduły

- Pozwalają podzielić złożoną aplikację na odrębne moduły
- Mogą posłużyć do tworzenia przestrzeni nazw, aby zapobiec kolizjom nazw między różnymi, odległymi elementami aplikacji
- Wraz z rozdzieleniem zawartości na wiele plików, pozwalają w czytelny sposób uporządkować złożony kod

Zewnętrzne biblioteki

- Czasem potrzebujemy skorzystać z zewnętrznych bibliotek np. jQuery, Bootstrap, AngularJS
- Musimy dostarczyć do naszego kodu zestaw definicji typów dla obiektów biblioteki (coś w rodzaju pliku nagłówkowego dla C/C++)

https://github.com/DefinitelyTyped/DefinitelyTyped

Zewnętrzne biblioteki

```
declare module PIXI {
    export class Spine extends DisplayObjectContainer {
        constructor(url: string);
        createSprite(slot: Slot, descriptor: { name: string }): Sprite[];
        update(dt: number): void;
    }
}

declare function requestAnimFrame(callback: Function): void;

declare module PIXI.PolyK {
    export function Triangulate(p: number[]): number[];
}
```

Przykładowo dla Pixi.js musimy dostarczyć plik pixi.d.ts w odpowiedniej wersji

Sprawdźmy to w praktyce

Warsztat

Prosta gra HTML5 w TypeScript

Spotkania z TypeScript

Dzięki i do zobaczenia :)