TypeScript
Triedy a rozhrania

kurz Vývoj progresívnych webových aplikácií

Eduard Kuric

STU FIIT

VSUVKA

- Electron – rámec, ktorý umožňuje vytvárať desktopové GUI aplikácie, z FE a BE komponentov vyvinutých pre webové aplikácie
- Device Access
- Quasar – Electron build [UKÁŽKA]
  - `quasar dev -m electron`
  - `quasar build -m electron`
  - Build commands
- Quasar - Asynchrónne komponenty [UKÁŽKA]
  - Vue asynchrónne komponenty

Trieda - `class`

- Definícia triedy

```
class Person {
    id: string;
    firstName: string;
    lastName: string;

    constructor(id:string, firstName:string,
                lastName: string) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

Metóda v triede, inštancia

- Definícia metódy v triede

```
class Person {
…
 getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
 }
}
```

- Vytvorenie inštancie

```
let person = new Person('123-11','Peter', 'Falk');
person.getFullName()
```

Modifikátory prístupnosti

- `private, protected, public`
- prístupnosť je kontrolovaná v čase transpilácie (nie runtime)
- ak nie je uvedený žiaden modifikátor pri atribútoch a metódach triedy, potom je použitý implicitne `public`

Modifikátory prístupnosti /2

- `private` – viditeľnost z vnútra triedy
- `protected` – viditeľnost z vnútra triedy a z podtried (subclasses)
- `public` – viditeľnosť odkiaľkoľvek

Deklarácia a inicializácia zároveň

- priamo v konštruktore, kratší kód

```
class Person {
  constructor(protected id: string,
              private fullName: string) {
    this.id = id;
    this.fullName = fullname;
  }
}
```

Modifikátor `readonly`

- Atribúty sú nemenné (immutable)
- Priradenie hodnoty `readonly` atribútu je možné iba:
  - pri deklarácii atribútu
  - v konštuktore

```
class Person {
    constructor(readonly birthDate: Date) {
        this.birthDate = birthDate;
    }
}
```

`readonly` **vs** `const`

- `readonly`
  - atribúty v triedach/rozhraniach
  - inicializácia pri deklarácii alebo v konštruktore
- `const`
  - premenné
  - inicializácia pri deklarácii

Getters, setters

- Umožňujú nám mať pod kontrolou prístup k atribútom tried
- Tiež im niekedy hovoríme accessors/mutators

```
class Person {
 private _firstName: string;
 private _lastName: string;
 public get firstName() {
   return this._firstName;
 }
 public set firstName(theFirstName: string) {
  if (!theFirstName) { throw new Error('Invalid first name.'); }
  this._firstName = theFirstName;
 }
 public getFullName(): string {
  return `${this._firstName} ${this._lastName}`;
 }
}
```

Dedenie - inheritance

```
class Employee extends Person {
  constructor(
     firstName: string,
     lastName: string,
     private jobTitle: string) {
        super(firstName, lastName);
    }
}
```

Prekonanie metód - overriding

```
class Employee extends Person {
 constructor(
    firstName: string,
    lastName: string,
    private jobTitle: string) {
       super(firstName, lastName);
   }
   // uvazujme ze v Person je tiez metoda describe
   describe(): string {
     return super.describe() +
               `I'm a ${this.jobTitle}.`;
   }
}
```

Statické atribúty a metódy

- statické atribúty/metódy sú zdieľané naprieč všetkými inštanciami triedy

```
class Employee {
    private static headcount: number = 0;

    constructor(
        private firstName: string,
        private lastName: string,) {
        Employee.headcount++;
    }
}
```

- Vyskúšajte si ako sa správajú statické atribúty a metódy pri dedení.

Abstraktná trieda

- Z abstraktných tried **nemôže byť vytvorená inštancia**
- Abstraktná trieda musí mať **prinajmenšom jednu abstraktnú metódu**
- Na **použitie** abstraktnej triedy potrebujeme:
  - **zdediť** z abstraktnej triedy
  - **poskytnúť implementáciu** pre abstraktné metódy

Abstraktná trieda /2

```
abstract class Employee {
  constructor(private firstName: string,
              private lastName: string) {
  }
  // iba signatura metody
  abstract getSalary(): number
  get fullName(): string {
     return `${this.firstName} ${this.lastName}`;
  }
}
```

Rozhranie

- Definujú kontrakty v kóde, čo je v “zmluve”, to platí
- Poskytujú explicitné názvy na kontrolu typov
- Rozhranie môže mať *voliteľné* atribúty a `readonly` atribúty
- Rozhranie môže byť použité ako typ funkcie
- Zvyčajne sa rozhrania používajú ako typy pre triedy
  - slúži ako kontrakt medzi “nesúvisiacimi” triedami

Rozhranie – voliteľný atribút

```
interface Person {
    firstName: string;
    middleName?: string;
    lastName: string;
}
```

Rozhranie – `readonly` atribút

```
interface Person {
    readonly id: string;
    firstName: string;
    lastName: string;
}
let person: Person;
person = {
    id: '121-11',
    firstName: 'Peter',
    lastName: 'Falk'
}
```

Rozhranie – typ funkcie

```
interface StringFormat {
    (str: string, isUpper: boolean): string
}


let format: StringFormat;
format = function (str: string,
                   isUpper: boolean) {
  return isUpper ? str.toLocaleUpperCase()
                 : str.toLocaleLowerCase();
};
```

Rozhranie – typ triedy

```
interface Json {
    toJSON(): string
}

class Person implements Json {
    constructor(private firstName: string,
        private lastName: string) {
    }
    toJson(): string {
        return JSON.stringify(this);
    }
}
```

Rozhrania - dedenie

- Rozhranie môže **dediť z** jedného a viac rozhraní

```
interface A {
  a(): void
}

interface B extends A {
  b(): void
}
```

Rozhrania – dedenie /2

- Rozhranie môže **dediť** aj z triedy
  - Ak ale trieda obsahuje `private` členov, rozhranie je možné implementovať iba danou triedou, alebo nadtriedou danej triedy

```
class Control {
  private state: boolean;
}
interface StatefulControl extends Control {
  enable(): void
}
class Button extends Control implements
      StatefulControl {
    enable() { }
}
// error
class Chart implements StatefulControl {}
```