Training TypeScript

Module: Type Guards

ORBIT GT BELGIUM

Peter Kassenaar

info@kassenaar.com

“Type Guards allow you to narrow down the type of an object within a conditional block”

Two main types of guards:

- `typeof` operator (…/`18-type-guard-typeof.ts`)
- `instanceof` operator (…/`19-type-guard-instanceof.ts`)

If you want to – you can declare your own, custom Type Guards (`20-type-guard-user-defined.ts`)

Type Guard using typeof operator

```
// type-guard-typeof.ts
function foo(bar: string | number) {
  if (typeof bar === 'string') {
    // do something, we KNOW it is a 'string' value
    // For instance, we get intellisense on all string methods
    return bar.toUpperCase();
  }
  // HERE, TypeScript KNOWS it should be a number value,
  // because we handled the string value above
  return bar.toFixed(2);
}
```

More real life example – step 1

```
class Employee {
  constructor(public name: string, public age: string | number) {}
}

function getEmployeeAge(employee: Employee) {
  // HERE, we implement the Type Guard
  /////………
}

const employeeAgeFromString = getEmployeeAge(
  new Employee('Dirk', '29')
);

console.log(employeeAgeFromString);
```

Step 2 – implement the type guard

We want the age of an employee ALWAYS to be of type `number`

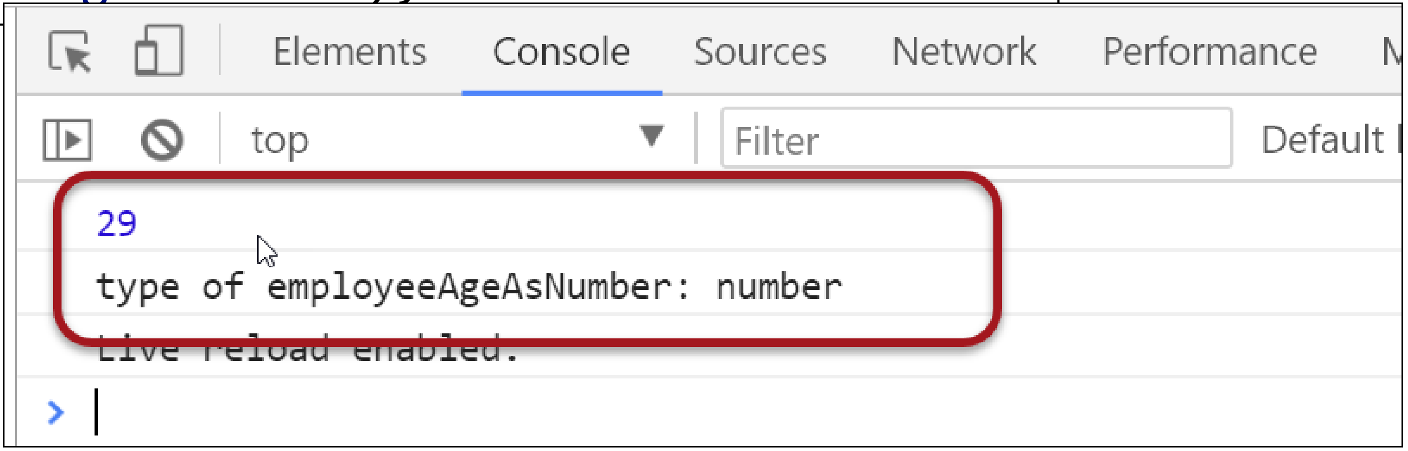
(but from an outside system (browser!), it might be passed in as a `string`).

So we write this helperfunction, using a type guard.

```
function getEmployeeAge(employee: Employee): number {
    // HERE, we implement the Type Guard
    if(typeof employee.age ==='number'){
        return employee.age; // simply return it. It’s already a number
    }
    return parseInt(employee.age); // convert to number, then return
}
```

Usage

```
const employeeAgeAsNumber =
    getEmployeeAge(new Employee('Dirk', '29'));

console.log(employeeAgeAsNumber);
console.log('type of employeeAgeAsNumber:',
    typeof employeeAgeAsNumber);
```



```
29
type of employeeAgeAsNumber: number
Live reload enabled.
```

We've had conversion functions in JavaScript for ages, but by using a Type Guard, we are SURE our parameters are of a certain type.

Instanceof Type Guards

What does `instanceof` actually do? It compares the `prototype` of two objects.

If they are the same, the objects are apparently derived from the same instance.

```
// 1. simple example: what does instanceof actually do?
class Foo {
  something() {}
}

const bar = new Foo();
console.log(bar instanceof Foo); // true
```

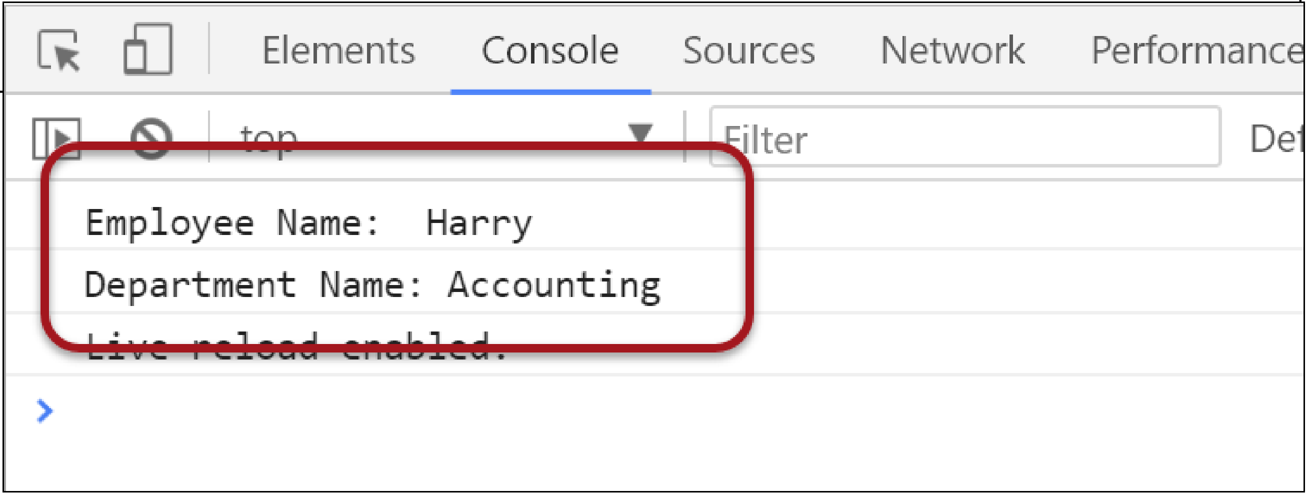
More real world example: get item name from a parameter

```
class Employee {
  constructor(public name: string, public age: string | number) {}
}

class DepartmentList {
  constructor(public title: string, public employees: Employee[]) {}
}
```

```
const employeeName = getItemName(new Employee('Harry', 52));
console.log('Employee Name: ', employeeName);

const departmentName = getItemName(
  new DepartmentList('Accounting', [
  new Employee('Astrid', 22),
  new Employee('Theo', 24)
  ])
);
console.log('Department Name:', departmentName);
```

and the function `getItemName()`… (looks overly complicated with multiple castings)

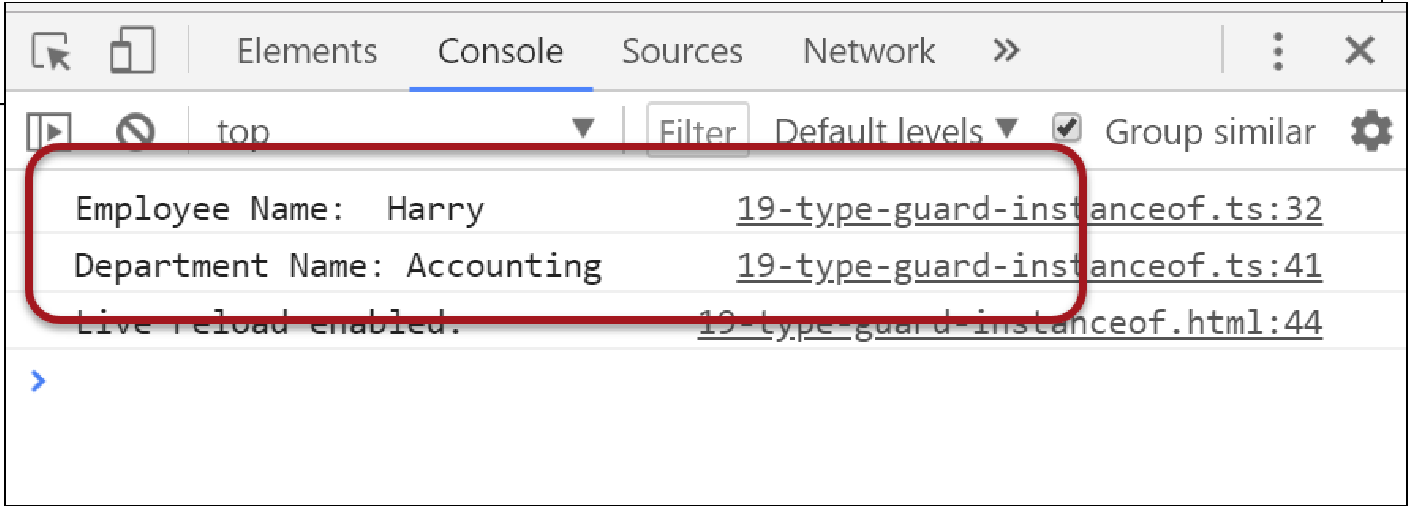
```
function getItemName(item: Employee | DepartmentList): string {
  //ugly solution, cast each item
  if ((item as Employee).name) {
    // apparently we're dealing with an Employee
    return (item as Employee).name;
  }
  // we're dealing with a DepartmentList
  return (item as DepartmentList).title;
}
```



```
Employee Name:  Harry
Department Name: Accounting
Live reload enabled.
```

Cleaner solution – use instanceof type guard

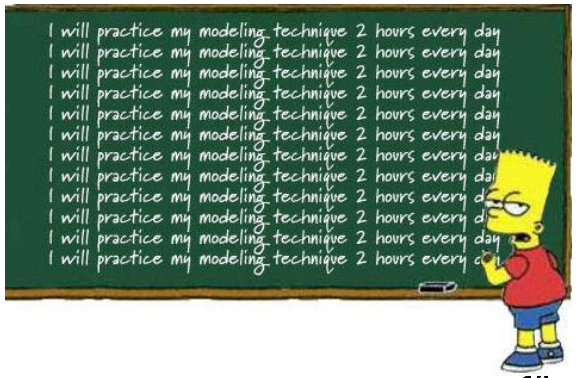
```
function getItemName(item: Employee | DepartmentList): string {
  // Nice solution, use instanceof operator
  if (item instanceof Employee) {
    // We're dealing with an Employee
    return item.name;
  }
  // we're definitely dealing with a DepartmentList
  return item.title;
}
```



```
Employee Name:  Harry            19-type-guard-instanceof.ts:32
Department Name: Accounting      19-type-guard-instanceof.ts:41
Live reload enabled.           19-type-guard-instanceof.html:44
```

Workshop

- Create a function with a Type Guard that accepts a number or an array.
  - If an array is passed in, it returns the length of the array
    - `[ 1, 2, 3]; // returns 3,`
  - If a number is passed in, it returns the sum of the individual numbers.
    - `491; // returns 14 (=4+9+1)`
    - See `../18-type-guard-typeof.ts` as example
- Create two classes, create an instanceof Type Guard to return one of the properties.
  - See `../19-type-guard-instanceof.ts` as example



User Defined Type Guards

- Create your own Type Guards by determining if a parameter is of a certain type: do so by creating a *Helper function*

```
function getItemName(item: Employee | DepartmentList): string {
  // Use custom function to determine if it is some type
  if (isEmployee(item)) {
    // We're dealing with an Employee
    return item.name;
  }
  // we're definitely dealing with a DepartmentList
  return item.title;
}

// Helper function: defer the type guard to a custom function
function isEmployee(item: any): item is Employee {
  return item instanceof Employee;
}
```

By adding the `item is Employee` as the return type, we are casting the boolean result of the `instanceof` comparison back to the desired type!

Example: `20-type-guard-user-defined.ts`